UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática


# ECOJUKEBOX

João Nuno Andrade Lobo

PROJETO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2012

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática


# ECOJUKEBOX

João Nuno Andrade Lobo

PROJETO

Trabalho orientado pelo Prof. Doutor Paulo Jorge Cunha Vaz Dias Urbano

e co-orientado pelo Prof. Doutor Francisco Cipriano da Cunha Martins

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2012

# Agradecimentos

Começo por agradecer à minha família, ao meu pai, à minha mãe e à minha avó pelo apoio incondicional ao longo da minha formação académica e que me permitiu chegar até aqui. Um agradecimento especial para a minha namorada que tem acompanhado estes últimos anos de formação e pela sua compreensão e apoio incondicional. Agradeço também a todos os meus amigos, colegas e conhecidos que me ajudaram e me fizeram acreditar que era capaz de chegar tão longe.

Finalmente quero agradecer ao Prof. Doutor Paulo Jorge Cunha Vaz Dias Urbano e ao Prof. Doutor Francisco Cipriano da Cunha Martins pelo excelente desempenho das suas funções de orientador e co-orientador, respetivamente. Agradeço todo o apoio prestado, disponibilidade e todos os ensinamentos que adquiri ao longo do desenvolvimento deste projeto.

*O destino é uma questão de escolha.*

# Resumo

Num ambiente social em que se pretende ouvir música, é importante tentar perceber dois aspetos fulcrais: quem fornece e quem seleciona as músicas a tocar. Neste contexto a *EcoJukebox* é vista como uma aplicação que permite a partilha de músicas entre vários dispositivos num único repositório comum e possibilita a interação das pessoas relativamente à decisão da escolha das músicas.

Este tipo de aplicação pretende criar uma maior competição/colaboração de todas os intervenientes nestes espaços sociais, permitindo uma participação mais ativa, através da atribuição de anotações às músicas e das reações a cada música tocada.

O método de decisão das músicas a tocar na *EcoJukebox* deve também ter em conta as diferentes situações públicas, por isso são importantes questões como o motivo do encontro social, a periodicidade, o tipo de grupo ou o local do evento.

O principal objetivo deste projeto é estudar várias formas de decisão na seleção de músicas com base no contexto, nos organismos de média existentes e na interação com as pessoas presentes. Foi também realizada uma avaliação do utilizador, que se baseou na: utilização do sistema, questionários e entrevistas para obter resposta dos utilizadores sobre os algoritmos e interação. Os resultados apresentam a *EcoJukebox* como uma plataforma agradável e divertida para a decisão em grupo, que motiva muito os utilizadores para a utilização de sistemas de reprodução de música em grupo e para a partilha de informação musical.

**Palavras-chave:** Música, Social, *Jukebox*, Decisão, Anotações

# Abstract

In a social environment where people want to listen to music, it is important to try to understand two principal roles: who provides and who selects the music to play. In this case, an application like *EcoJukeBox* is seen as an application that allows sharing music between multiple devices into one common repository and allowing the interaction of people in roles to decide which songs to play.

This type of application wants to create a larger competition/collaboration among all the users of these social spaces, allowing a more active participation, by assigning tags to the songs and reactions to played music.

The decision method of the music playing on the *EcoJukeBox* must take into consideration the different public goal scenarios. Issues like the reasons for the social meeting, its periodicity, the kind of group, or the location of the event play an important role in the way *EcoJukeBox* may react.

The prime goal of this project is to study different ways music may be selected depending on the social context, in the existent media, and in the interaction among *EcoJukeBox* listeners. Was also performed a user evaluation consisting in: system usage, questionnaires and open interviews to collect user feedback about the algorithms and interaction. The results present *EcoJukeBox* as an enjoyable and entertaining group decision platform which highly motivates users.

**Keywords:** Music, Social, Jukebox, Decision, Tags

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

A música é um fenómeno social, que consegue captar a atenção das pessoas pelas emoções que nelas provoca. É por isso motivo de variadíssimos estudos que pretendem registar os comportamentos das pessoas em grupo e de que forma se relacionam.

O conceito de *Jukebox* surgiu associado pela primeira vez ao fonógrafo, uma invenção de *Thomas Edison* [1], o qual pretendeu revolucionar a indústria com um aparelho inovador que permitia a gravação de voz. Mas a *Jukebox* viria a ser muito mais do que isso. Foi essencialmente um sucesso por ser um meio de entretenimento inovador que permitia o armazenamento em massa de música e ainda mais importante, introduziu o conceito de música social, pois era tocada normalmente em zonas públicas, tais como bares.

Decorrente do avanço tecnológico constante e rápido, atualmente, o conceito original de *Jukebox* apresenta algumas limitações em relação às necessidades atuais da sociedade. Assim sendo, é importante ter em conta as novas formas de interação social existentes, onde os dispositivos móveis têm um papel relevante, bem como os dispositivos de reprodução de áudio/musica. É também necessário melhorar a tecnologia existente para fomentar o interesse da música nas novas gerações e chegar mais facilmente a um maior número de pessoas.

Neste capítulo é descrita a motivação que está na base da elaboração do presente projeto, isto é, quais os principais fatores que culminaram no desejo de o concretizar, bem como os objetivos definidos para alcançar com a realização do projeto. São também apresentadas as suas contribuições e por fim é descrita a estrutura do documento.

Este trabalho foi realizado no contexto do Projeto em Engenharia Informática, para o Mestrado em Engenharia Informática, na Faculdade de Ciências da Universidade de

Lisboa, no grupo de investigação do Departamento de Informática *LabMAg - The Laboratory of Agent Modelling*.

## 1.1 Motivação

A *Jukebox,* apesar de já ter sido inventada há muitos anos, reúne ainda um número considerável de seguidores e apreciadores do conceito. No entanto, é necessário cativar novos públicos, proporcionando um contributo social importante e tirando partido de uma sociedade cada vez mais recetiva a produtos inovadores. Para isso é necessário ter em conta muitos fatores que irão influenciar a escolha das músicas que irão ser tocadas. É necessário identificar o contexto em que a *Jukebox* vai ser utilizada, pois diferentes situações públicas requerem diferentes tipos de decisão, tais como festas, espaço de lazer numa escola, estação de metro ou um consultório médico. Podem ser ambientes familiares em que o grupo de pessoas se repete, ou ambientes em que as pessoas aparecem esporadicamente. Ainda importa também ter em consideração o carácter do evento, pois este varia consoante seja periódico, permanente ou ocasional.

Se combinarmos todos os estes fatores na decisão das músicas a tocar, associada a uma maior interação, competição e colaboração por parte das pessoas nessa mesma decisão, naturalmente a experiência de ouvir música através de uma *Jukebox* irá ser muito mais entusiasmante, enriquecedora e necessariamente terá muitos mais seguidores.

## 1.2 Objetivos

Com este projeto pretende-se construir uma aplicação que simule o funcionamento de uma *Jukebox*, permitindo a interação com diferentes dispositivos, fixos ou móveis, para construírem a lista de músicas disponíveis para tocar. A aplicação *EcoJukeBox* é um sistema *online* de partilha de músicas, uma forma de ouvir música em grupo e interagir na decisão da escolha das mesmas.

É importante portanto traçar um perfil do grupo a que se destina a música. Para isso, neste trabalho é criado o conceito de critério. Critérios são preferências ou exigências do grupo que vai ouvir a *EcoJukeBox*. São portanto os ouvintes a decidir uma série de filtros que possam melhorar a decisão de músicas de acordo com a maioria das preferências do grupo.

Em relação a cada interveniente do evento, importa possibilitar que interajam com a aplicação como parte integrante na escolha das músicas, dando-lhes a possibilidade de expressar opiniões, demonstrar os seus gostos musicais e até formarem grupos com outras pessoas no evento com as mesmas preferências, com o objetivo de "dominarem" a *Jukebox*.

## 1.3 Contribuições

Neste projeto é proporcionada uma experiência de música em grupo. Todos os utilizadores podem disponibilizar os seus próprios ficheiros de música numa pasta partilhada do tipo *SugarSync*. A música existente nessa pasta partilhada pode vir a ser tocada na *EcoJukeBox*, permitindo que cada pessoa possa contribuir de forma ativa na escolha das músicas que possam vir a tocar. Os utilizadores podem aceder a várias informações através de um navegador, onde encontram informações sobre a música que está a tocar, sobre os álbuns existentes, as anotações, etc. Podem ainda influenciar as decisões da *EcoJukeBox* através de votações em álbuns, anotações, gostos e não gostos na música que está a tocar no momento. É também permitido aos utilizadores associarem novas anotações às músicas. A *EcoJukeBox* gere a informação relativa às músicas, álbuns, anotações e votações guardando-as numa base de dados. Disponibiliza ainda uma série de serviços na *Web* que vão ser usados quer para enviar informações para o navegador, quer para receber os resultados da interação com os utilizadores e guardá-los igualmente numa base de dados. É também possível criar perfis de grupo na *EcoJukeBox*, onde é descrito um conjunto de critérios a que se recorre para decidir qual a música que vai tocar a seguir num determinado espaço comum. Esses critérios podem ser exigências ou preferências.

## 1.4 Estrutura do documento

O relatório encontra-se estruturado em cinco capítulos para além desta introdução. No capítulo seguinte (capítulo 2) é descrito o trabalho relacionado com o presente projeto. No capítulo 3 será explicado o funcionamento da *EcoJukeBox*, enquanto que no capítulo 4 será descrito o desenho do sistema. A avaliação da *EcoJukeBox* será abordada no capítulo 5 e a discussão e trabalho futuro é exposta no capítulo 6.

# Capítulo 2

# Trabalho Relacionado

A música social não é um novo conceito, e existe hoje em dia uma grande comunidade que se dedica a estudar este problema e a tentar integrá-lo com a computação artificial. O problema permanece em aberto, existindo várias abordagens estudadas, que destacamos de seguida.

O projeto *Jukola* [2] é um desses trabalhos já desenvolvidos na ótica da música social. Neste é descrito um leitor de *MP3* interativo, criado para ser usado por um grupo de pessoas num espaço público. A escolha de músicas é feita democraticamente, ou seja, existe um monitor táctil com uma lista de músicas, localizado num espaço público (bar, universidade, etc.). Este monitor é usado para os utilizadores votarem na música que querem que seja tocada de seguida. Como alternativa, também existe uma versão móvel permitindo aos utilizadores votarem na próxima música através do *PDA* usando uma rede sem fios. As bandas e os artistas que querem ver a sua música a tocar nesse espaço têm a possibilidade de carregar os seus ficheiros *MP3* através da *Internet*.

Este projeto *Jukola* tem algumas semelhanças com o projeto *EcoJukeBox*, principalmente no que se refere à partilha de ficheiros através da *Web*. Relativamente à escolha de músicas, o projeto *Jukola* apenas se baseia nas votações em músicas por parte dos utilizadores, enquanto que neste projeto a escolha de músicas tem em conta votações em álbuns, anotações, gostos e não gostos na música que está a tocar no momento.

Na sequência do projeto *Jukola*, surgiu o projeto *PartyVote* [3], que usa o conceito de voto do *Jukola* e aplica-o a situações de grupo. Trata-se de uma *Jukebox* de música democrática que pretende dar a todos os participantes a mesma influência na música tocada em situações sociais. Foi também projetado para estabelecer grupos sociais sem nenhum perfil de utilizador pré-existente e fornecendo a música adequada para esse

grupo. A visualização usa redução de dimensionalidade[1] para mostrar as semelhanças entre as músicas e de que forma a sobreposição de votos influencia a música tocada. Visualizar as decisões de voto permite ligações a outros indivíduos, proporcionando a formação de grupos com os mesmos interesses.

Ao contrário deste trabalho, neste projeto as decisões de voto apenas são utilizadas internamente para decidir a próxima música a tocar. Uma das semelhanças deste projeto com o *PartyVote* é o facto de não haver registos de utilizadores, havendo apenas a preocupação de fornecer a música adequada para um grupo de pessoas.

Outro estudo sobre a seleção de música em ambientes sociais é descrito no artigo *Exploring Social Music Behavior: An Investigation of Music Selection at Parties* [4]. Neste trabalho reflete-se sobre como o tamanho dos grupos influencia a forma como a música é ouvida e compreendida. Este artigo descreve algum *software*/*hardware* que permite selecionar e ouvir música. Outro dos pontos abordados é o problema da seleção colaborativa de música. Os autores do artigo dividem os participantes destes eventos em dois tipos distintos: o anfitrião e os convidados. O anfitrião é responsável por criar a *playlist* inicial, enquanto que os convidados também têm um papel ativo, contribuindo com as suas músicas para a coletânea da festa e podem adicionar, apagar ou reordenar músicas da *playlist*.

No projeto *EcoJukeBox* foi usado o conceito de anfitrião e convidados com outro sentido. O anfitrião é responsável por criar o perfil de grupo, decidindo em conjunto com todos os utilizadores do sistema os critérios de seleção das músicas. Os utilizadores desta aplicação têm uma maior influência na escolha das músicas por intermédio das votações na interface *Web*.

A aplicação *Push!Music* [5] é usada em dispositivos móveis sem fios. Os ficheiros de música tomam a forma de agentes de *software* autónomos, que constroem a sua própria personalidade à custa de meta-dados de outros agentes que encontram. Têm autonomia para se movimentar autonomamente através dos dispositivos móveis na proximidade, procurando o ambiente que mais lhes convenha. Os utilizadores da aplicação podem fazer as suas próprias recomendações através de partilha colaborativa ou *pushing* de música para outros utilizadores na vizinhança.

[1] Transformação de uma grande quantidade de dados, num conjunto reduzido de características mais representativas

Este estudo é interessante, pois ao contrário deste trabalho que se centra nas escolhas dos utilizadores para tomar decisões, centra-se nos próprios ficheiros de música e na sua própria personalidade.

Outro estudo importante é abordado no artigo *Extending the folksonomies of freesound.org using content-based audio analysis* [6], relativamente ao uso de anotações. É feito um estudo aprofundado dos mecanismos sociais de anotação usados em *freesound.org*, uma comunidade *online* onde os utilizadores podem navegar em ficheiros de áudio através de anotações e fazer pesquisas de áudio baseando-se na semelhança de conteúdo. Neste estudo são analisados vários problemas já conhecidos nos sistemas de anotação, tais como: polissemia, sinonímia e a escassez de anotações existentes. É também proposto um "autotag" das músicas, através de um classificador *ak-NN*[2], que seleciona as anotações disponíveis a partir dos sons mais similares, para tentar resolver o problema de músicas com falta de anotações.

Na *EcoJukeBox* as anotações também são muito utilizadas, existindo a possibilidade de adicionar novas anotações às músicas e também de votar em anotações. No entanto não se foi tão longe no uso de anotações, pois não existe neste trabalho nenhuma forma de "autotag" de músicas.

Ainda em relação à criação de *playlists* é importante referir o trabalho desenvolvido no artigo *Automatic Playlist Composition in a Dynamic Music Landscape* [7], que sugere a música como uma experiência não apenas social, mas também pessoal. Apesar da satisfação geral dos utilizadores com os resultados produzidos, os mesmos pretendem uma maior capacidade de interação e uma maior expressividade semântica. O objetivo será combinar as expectativas pessoais dos ouvintes com a experiência dos compositores profissionais. Sugere-se portanto que a composição de *playlists* não deva ser apenas da exclusiva responsabilidade dos computadores, exigindo também um alto nível de interação humana, para uma experiência mais satisfatória para os utilizadores.

Outro trabalho interessante, *Pulling Strings from a Tangle: Visualizing a Personal Music Listening History* [8] refere-se ao uso do historial de músicas que cada pessoa ouviu, como uma peça importante para a escolha de novas *playlists*. Neste artigo, apresentam três conjuntos de dados diferentes, com os quais são geradas novas *playlists* a partir do historial de um utilizador. Apresentam três visualizações para um historial de música pessoal, um representando semelhança entre músicas baseado em proximidade

[2] Classificador do vizinho mais próximo

temporal e espacial entre os utilizadores, e os outros dois descobrindo padrões que destacassem as sessões de escuta.

Na *EcoJukeBox* foi guardado todo o histórico com as interações entre os utilizadores e o sistema e com as músicas que iam sendo adicionadas e tocadas. No entanto, esse historial tinha apenas o objetivo de poder estudar de que forma o algoritmo da escolha de músicas se comportava e de que forma poderia ser melhorado.

Um dos projetos desenvolvidos mais recentemente nesta área foi o *GroupFun* [9]. Trata-se de uma aplicação *Web* que ajuda um grupo de amigos a chegar a um acordo sobre a escolha de uma lista de música comum para um determinado evento. É um aplicativo de música que permite aos indivíduos organizar e compartilhar as suas músicas favoritas com grupos. Em *GroupFun* os utilizadores podem ouvir a sua própria coleção de músicas, bem como a música dos seus amigos. Com uma base de dados musical coletiva, o aplicativo integra os gostos musicais dos amigos e recomenda uma lista comum para eles. Portanto, o aplicativo tem como objetivo satisfazer os gostos musicais de todo o grupo através da agregação de preferências individuais.

A principal contribuição do *GroupFun* centra-se na avaliação de quatro algoritmos, determinísticos ou probabilísticos, que têm em consideração o peso de cada música. Foi debatido o papel das preferências dos membros em termos de familiaridade e descoberta de novas músicas num cenário de tomada de decisão. Os resultados apontaram para o fato de que a incorporação de uma distribuição de probabilidades de classificações nas músicas aumentaria a diversão em grupo e surpreenderia os membros com excelentes recomendações provenientes de seus amigos. Mecanismos de agregação de preferências incorporam pesos e fatores sociais que devem ser investigados com o objetivo de aumentar a aceitação da decisão e a diversão em grupo.

A aplicação *MEET* [10] apresenta uma abordagem para interfaces distribuídas como tecnologia de suporte à interação social. É constituída por um sistema de música colaborativo, que para além de beneficiar de armazenamento de música distribuída, também distribui controlo de reprodução em vários dispositivos móveis e fornece saída através de um visor comum. A ideia principal do *MEET* é permitir que os utilizadores possam influenciar a música tocada em dois níveis: primeiro, permitindo aos utilizadores compartilhar músicas selecionadas da sua coleção pessoal de música digital, segundo, permitindo que o grupo de utilizadores controle a reprodução conjuntamente através de uma interface distribuída. A interação é, portanto, espalhada pelos vários dispositivos, cada um com a sua função distinta. O conceito desenvolvido neste trabalho

surge do resultado de uma análise de tecnologia, onde os conceitos que não estão a ser explorados nos sistemas atuais foram identificados. Como resultado chegou-se a um conjunto de orientações gerais para a interação do sistema: a música nunca pára, nunca é interrompida e há sempre uma nova pronta para tocar. O sistema é secundário, ou seja, não foi projetado para ser o centro das atenções, mas sim um sistema de música tocando apenas como música ambiente. Também é possíveis diferentes níveis de participação. É possível nomear músicas, que assim se tornam candidatas a serem tocadas. Nesse caso, essa canção é incluída num mostrador que indica as músicas disponíveis para votação e permanece lá até ser tocada ou eliminada. Caso não haja pelo menos três músicas indicadas para votação o sistema escolhe de forma aleatória do sistema para completar no mostrador de votações. Isso irá assegurar que há sempre uma música pronta para ser tocada e, além disso, incentivar a interação dos utilizadores por haver sempre músicas para votar. Outra característica importante do sistema é que nenhuma pessoa pode decidir apenas por ela quais as músicas a ouvir, apenas sendo permitido votar nas músicas nomeadas por outros. A interface de votação apresenta aos utilizadores uma lista de nomeações onde podem votar em cada candidato. Um voto positivo ou negativo simplesmente adiciona ou subtrai um ponto a partir de uma pontuação total, respetivamente, e a música com a maior pontuação será tocada de seguida. São adicionadas regras de eliminação que são aplicadas no final de cada música, para tornar o sistema mais dinâmico. A primeira regra remove candidaturas com uma pontuação total negativa. A segunda regra remove nomeações que não receberam votos (positivo ou negativo), após três músicas. O sistema foi testado em três diferentes contextos de vida real, a fim de explorar o espaço de interação. A intenção de ter três ensaios não foi de replicar a experiência, mas sim de experimentar o sistema em diferentes configurações físicas e contextuais. Os resultados apontaram complicações dos utilizadores, em dar atenção distribuída por diferentes dispositivos, bem como questões relativas a monitores comuns para obter resposta de vários utilizadores.

Este projeto é semelhante à *EcoJukeBox* no seu conceito, na partilha de músicas e na participação dos utilizadores, mas difere na medida em que na *EcoJukeBox* não existe o conceito de votação apenas nas músicas sugeridas por outros. Este projeto também não refere o uso de anotações.

No projeto *This is a Radio Bot* [21], desenvolvido pelo colega Diogo Serrano, foi criada uma plataforma de rádio *Web*, que permite aos utilizadores assistirem a diversos programas preparados por diversos *Djs* artificiais. A gestão da rádio é feita por um painel de administração, que decide os horários e duração dos vários programas de rádio, bem como decidir a personalidade e gosto musical dos *Djs* artificiais que vão

criar as playlists para os diversos programas de rádio. Para que as músicas pudessem chegar aos ouvintes, foi criado um *Player* e um servidor de rádio. O *Player* é o responsável por executar os programas de rádio definidos anteriormente no painel de administração.

Na *EcoJukeBox* reutilizam-se alguns dos recursos e tecnologias usadas neste trabalho: a forma como as músicas são persistidas para a base de dados, recorrendo a JPA; a utilização do mesmo servidor de aplicação, o *Glassfish* [24] [glossário: G]; o recurso à *Last.Fm* para obter informação complementar sobre as músicas; e o uso do *Player* que faz *streaming* das músicas para um servidor de rádio, modificando-o para receber uma música de cada vez (a música escolhida para tocar de seguida), em vez de receber *playlists* de música relativas a programas de rádio.

# Capítulo 3

# EcoJukeBox

A *EcoJukeBox* é um sistema *online* de partilha de músicas, uma forma de ouvir música em grupo e interagir na decisão da escolha das mesmas. Todos os utilizadores podem colocar os seus conteúdos musicais preferidos, de uma forma anónima, num único repositório, que se trata de uma pasta partilhada do serviço *SugarSync* [20] [glossário: G]. A *EcoJukeBox* gere a informação sobre as músicas que são colocadas nesse repositório, guardando-as numa base de dados e recorre à *Last.FM* [11] para complementar os dados relativos às músicas que são extraídas dos respetivos ficheiros *MP3* [glossário: G].

Os utilizadores são anónimos, mas partilham um perfil, a que damos o nome de perfil de grupo. A este perfil estão associados vários critérios para a escolha de músicas, os quais influenciam a música que vai ser selecionada para tocar. Podem ser definidos vários critérios, chamados de exigências, que inibem músicas de tocar no sistema. Podem também ser definidas várias preferências musicais que vão influenciar a escolha das músicas na *EcoJukeBox*. Este perfil é decidido por todos os elementos do grupo, os quais se reúnem para definir os critérios.

É disponibilizada uma forma de interação com o sistema através de uma interface *Web* que permite ouvir a música que está a passar na *EcoJukeBox* e também votar em anotações, músicas ou álbuns para influenciar a escolha das próximas músicas. Relativamente às votações nas músicas, os utilizadores podem fazê-lo sob a forma de "gosto" ou "não gosto" para expressarem a sua opinião em relação às mesmas.

A cada música está associada uma pontuação que indica o grau de preferência dos utilizadores. Sempre que uma música é escolhida para tocar, é necessário recalcular as pontuações das músicas. A todas está associada uma pontuação, quer satisfaçam ou não

as exigências. No caso de uma música não satisfazer as exigências não vai ser considerada para tocar, independentemente da sua pontuação.

Existem também vários eventos que podem ocorrer durante a execução da *EcoJukeBox*. Os eventos são importantes para identificar as várias formas de interação com o sistema, persistir a informação da sua ocorrência na base de dados e usar essa informação para aplicar critérios de seleção das músicas baseados em informação dinâmica.

Quando uma música acaba de tocar é escolhida uma nova que corresponde à que tem melhor pontuação no que diz respeito às preferências. Os critérios podem ser simples booleanos, no caso das exigências, como por exemplo, uma música ser de um determinado artista. Caso o critério seja uma preferência, por exemplo, uma música ter sido tocada recentemente, calcula-se um valor normalizado entre 0 e 1. Tipicamente uma música que tocou há pouco tempo neste caso terá um valor próximo de 1 e uma música que já não toca há muito tempo terá um valor próximo de 0.

## 3.1 Repositório de músicas

Tendo por base a análise de artigos que abordam a temática da música social em grupo, um das razões mais importantes que são apontadas como justificação para a utilização deste tipo de sistemas é o facto de os utilizadores poderem partilhar as suas músicas favoritas com outras pessoas. Os utilizadores que participam num evento deste género gostam de mostrar aos outros as suas escolhas musicais e querem essencialmente que as suas músicas toquem e sejam um sucesso. Tentam também com isso ganhar popularidade no grupo, tornando-se conhecidos pelo seu bom gosto musical. Para isso, influenciam os outros utilizadores a votar nas suas músicas e a sua popularidade começa a aumentar.

Neste trabalho existe um espaço comum de partilha de música, onde cada utilizador pode colocar as músicas que mais gosta e que assim se tornam candidatas a tocar na *EcoJukeBox*. Para isso é usado o serviço *SugarSync*, uma pasta partilhada em que os utilizadores têm acesso, fazem o *upload* das suas músicas favoritas e todas essas são automaticamente sincronizadas num único repositório de ficheiros musicais. É a partir desta pasta partilhada, que o sistema vai obter informação sobre as músicas, quer através dos meta-dados do ficheiro (*ID3 Tags*), quer através do próprio nome do ficheiro *MP3*. Irá ainda obter informação complementar através da *Last.Fm*, um serviço externo onde se pode obter dados sobre cada música e posteriormente persisti-los na

base de dados da *EcoJukeBox*. A Figura 3.1 ilustra o circuito que o sistema percorre na obtenção de informação musical e armazenamento na base de dados.

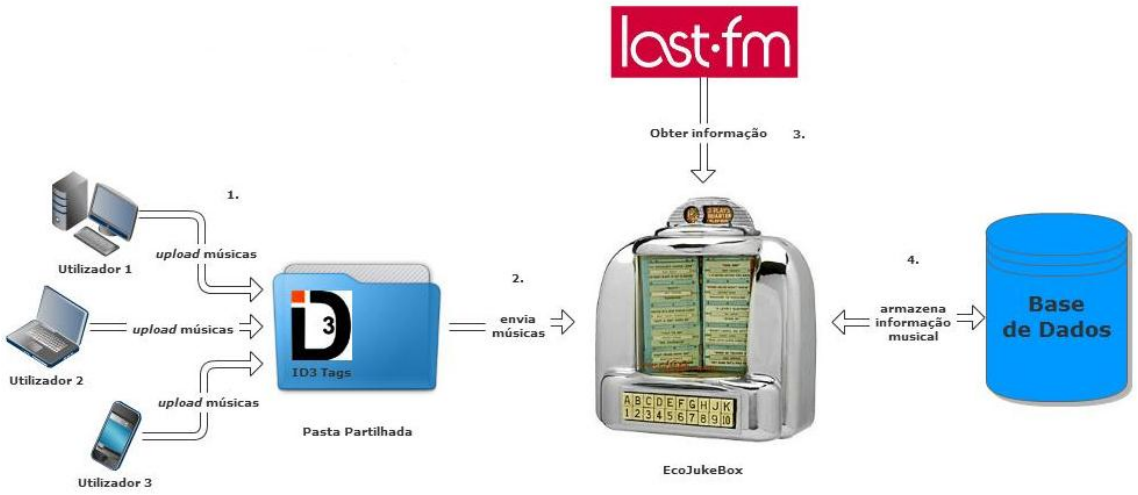



**Figura 3-1: Esquema relativo à obtenção de informação musical na *EcoJukeBox* e armazenamento na base de dados**

## 3.2 Persistência de informação

O conteúdo da informação guardada pode ser dividido em estática é dinâmica.

### 3.2.1 Informação Estática

A maior parte da informação guardada na base de dados refere-se a informação estática. É obtida quando a música é adicionada e não sofre alterações durante o funcionamento da *EcoJukeBox*. Esta informação refere-se essencialmente a todos os dados sobre as músicas que são obtidos, quer através dos meta-dados do ficheiro *MP3* (*ID3 Tags*), quer por meio da informação complementar obtida na *Last.Fm*. Na base de dados podemos encontrar informação estática referente às músicas (título, artista, duração, localização do ficheiro, género musical, data em que foi adicionada), às anotações (nome) e aos álbuns (nome, capa).

### 3.2.2 Informação Dinâmica

A informação dinâmica diz respeito aos dados que sofrem alterações decorrentes da interação com utilizadores. Estas alterações influenciam o processo de escolha das músicas na *EcoJukeBox*. Em relação à música é registado o número de vezes que tocou, a última vez que tocou, a sua pontuação, se a pontuação dessa música terá de ser

recalculada porque ocorreu um evento sobre ela, satisfaz ou não as exigências, e é registado o número de "gostos" e "não gostos". Também são guardadas as votações em álbuns e em anotações.

## 3.3 Interface com o utilizador

Um outro aspeto importante neste tipo de aplicações é a interação entre o utilizador e o sistema. O utilizador pretende ter um papel ativo na escolha das músicas a tocar e quer participar nesse processo de seleção. Para tal, as pessoas que utilizam sistemas de música social podem recorrer a estratégias para alcançar os seus objetivos, como por exemplo aliarem-se a outros participantes. Assim, o que vai ser tocado na sala são músicas selecionadas por si e outras que não sendo selecionadas por si vão ao encontro dos seus gostos musicais.

É deste modo importante manter um nível adequado de interação com o utilizador, ou seja, que o faça sentir relevante na escolha das músicas, garantindo que os utilizadores continuem a usar o sistema e a ouvir música através dele. Neste trabalho é usada uma interface *Web* (ver apêndice D), responsável por reproduzir a música escolhida pelo sistema, mostrar informações e permitir a interação do utilizador com o sistema. Nesta interface é possível ouvir a música que está a ser tocada na *EcoJukeBox* e conhecer a informação sobre essa música, desde título, autor, nome do álbum, capa do álbum e anotações associadas. Permite ainda a votação em álbuns, anotações, músicas através de "gostos" e "não gostos" e adição de novas anotações à música que está a tocar.

### 3.3.1 Reprodução de música

O sistema foi pensado para ser utilizado em diferentes contextos sociais, entre os quais a reprodução de música num espaço comum de trabalho. Foi pensado por exemplo para escritórios, em que os trabalhadores possam ter acesso a uma rede comum e em que cada utilizador possa ouvir música no seu computador pessoal, mas que permita uma partilha entre os colegas. Baseado nesta necessidade, a interface *Web* permite a reprodução de músicas recorrendo ao servidor de rádio *icecast2* [23] [glossário: G] para o qual é emitida a música que foi escolhida para tocar.

No projeto "This is a RadioBot" da autoria de Diogo Serrano [21], foi desenvolvido um *Player* responsável por fazer *streaming* das músicas para o servidor de rádio, que é usado neste projeto.

### 3.3.2 Informação estática sobre a música

Uma das informações que deve constar na interface *Web* diz respeito à música que está a tocar no momento. Essa necessidade prende-se com uma questão de contextualização e também permite ao utilizador saber mais alguma informação sobre a música, podendo relacionar e utilizar esse conhecimento numa próxima interação com o sistema.

Nesta interface é incluída informação sobre o artista da música atual, o título, as anotações associadas a essa música, a imagem com a capa do álbum e o nome do álbum.

### 3.3.3 Votação de músicas

A votação "gosto" ou "não gosto" na música atual é uma forma simples de obter informação de retorno por parte dos utilizadores em relação a uma música específica. Podem ainda manter um papel neutro não votando. O objetivo desta funcionalidade é poder associar-lhe critérios para que as músicas preferidas dos utilizadores possam passar mais vezes, enquanto que as músicas que os utilizadores não gostaram de ouvir não voltem a tocar na *EcoJukeBox*.

### 3.3.4 Votação de Anotações

As votações em anotações pressupõem que a escolha de músicas no sistema tenha em conta um determinado estilo que agrade aos utilizadores. As pessoas podem votar através desta interface *Web* em anotações que representem os seus gostos musicais. Por exemplo, uma pessoa que goste de música *rock* poderia votar na anotação *rock*. A interface *Web* disponibiliza as vinte anotações mais votadas numa *Tag Cloud*, em que as que têm mais votos são visualizadas com maior destaque em relação às anotações com menos votos. Dado que seria incomportável exibir todas as anotações, apenas são apresentadas algumas. A Figura 3.2 ilustra a interface usada para a votação de anotações na página *Web* deste trabalho.


**Figura 3-2: *Tag Cloud* usada na interface *Web***

Quando um dos critérios para a escolha da música é baseado em anotações, verifica-se o número de votos dos utilizadores numa anotação, recorrendo à base de dados. Obtido o número de votos, as músicas que tenham essa anotação vão subir a sua pontuação. O valor incrementado na pontuação de cada música corresponde ao número total de votos na anotação a dividir pelo número de músicas a que essa anotação pertence, repetindo a votação pelas músicas com essa anotação.

### 3.3.5 Votação de Álbuns

A votação num álbum é outra forma dos utilizadores influenciarem as escolhas das músicas. Ao votarem num determinado álbum, todas as músicas deste álbum são alvo de valorização e aumentam a possibilidade de virem a ser tocadas na *EcoJukeBox.* A interface *Web* disponibiliza uma lista com todos os álbuns que existem na *EcoJukeBox*. Estes álbuns normalmente não estão completos, ou seja, nem todas as músicas desse álbum fazem parte da base de dados musical do sistema. Esta referência a álbuns serve apenas para relacionar as músicas existentes na *EcoJukeBox* com os álbuns a que pertencem. A ideia de relacionar músicas do mesmo álbum é permitir que um utilizador possa identificar uma música que gosta, verificar o álbum a que pertence, e caso haja mais músicas do mesmo álbum no sistema, mostrar interesse em escutar estas músicas. Os critérios baseados em álbuns têm o mesmo comportamento do que o que foi referido anteriormente para os critérios de anotações.

### 3.3.6 Adição de Anotações às Músicas

Neste projeto, aquando da adição de uma música, recorre-se a informação complementar da *Last.Fm*, um serviço externo que se baseia precisamente neste conceito de anotações, geridas pela comunidade dos seus utilizadores, permitindo a descrição textual de uma música. No entanto, nem sempre as anotações que estão associadas a uma música são as mais corretas e por vezes faltam outras importantes que podiam caracterizar melhor essa música ou um conjunto de músicas. Para colmatar essa falha decidiu-se implementar uma forma de atribuir novas anotações a músicas. Enquanto uma música estiver a tocar na *EcoJukeBox* é permitido associar-lhe novas anotações que serão guardadas também na base de dados.

## 3.4 Definição de critérios para uma sala

Os ambientes sociais em que se pretende ouvir música são bastante diversificados. Esta aplicação pode ser usada por exemplo, num escritório, no cais do metro, num consultório médico ou num *open space* de um jornal. Este trabalho tem como objetivo conseguir adaptar-se às mais variadas situações e espaços onde se pretenda ouvir música em grupo. Tendo em conta a diversidade a que se poderá destinar esta aplicação, foi introduzido o conceito de perfil do espaço comum musical. Para cada espaço/grupo em que a *EcoJukeBox* irá tocar, o algoritmo de escolha de músicas terá em conta o perfil do grupo, que vai ao encontro das necessidades e preferências dos intervenientes em cada situação específica.

### 3.4.1 Exigências e Preferências

Para ser decidida qual a próxima música a ser tocada num determinado espaço comum a *EcoJukeBox* recorre a um conjunto de critérios que está associado a esse espaço, ou seja, ao seu perfil. Existem dois tipos de critérios: exigências e preferências. As exigências determinam se uma música pode ser selecionada para tocar. Caso uma música não satisfaça todos os critérios que são indicados como exigência daquele espaço comum, essa música não irá tocar. Depois de filtrar todas as músicas que podem ser tocadas, uma música terá maior classificação conforme o número de critérios preferenciais que conseguir cumprir. As preferências são condições que ajudam a decidir na escolha das músicas, mas não inibem nenhuma música de tocar.

Neste projeto foram implementados alguns critérios que podem ser usados para caracterizar um espaço social comum. No entanto, é possível implementar novos critérios que ainda não tenham sido contemplados. Os critérios podem também dividir-se no que se refere ao conteúdo da informação: baseados em informação estática e baseados em informação dinâmica.
A Figura 3.3 ilustra as relações entre os diferentes tipos de critérios. Como podemos verificar, os critérios baseados em informação dinâmica apenas são usados como preferências, enquanto que os critérios baseados em informação estática podem ser usados como preferências ou como exigências.

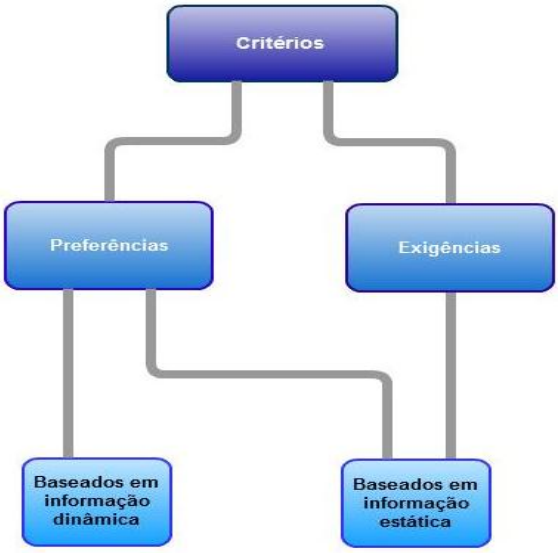



**Figura 3-3: Relações entre os diferentes tipos de critérios**

### 3.4.2 Critérios baseados em informação estática

Os critérios baseados em informação estática são normalmente usados para filtrar estilos de música, de artistas ou de álbuns. Todos estes critérios são booleanos e podem ser preferências ou exigências. Se por exemplo, ao traçar o perfil de um espaço musical, for observado que as pessoas intervenientes não gostam de música *pop*, convém incluir um critério que tenha em conta essa preferência musical do grupo. No projeto desenvolvido foram implementados uma série de critérios que podem ser aplicados às músicas deste género, nomeadamente:

- A música ser de um determinado artista;
- A música ter uma determinada anotação;
- A música ser de um determinado álbum;
- A música não ser de um determinado artista;
- A música não ter uma determinada anotação;
- A música não ser de um determinado álbum.

### 3.4.3 Critérios baseados em informação dinâmica

Os critérios baseados em informação dinâmica são a chave do sucesso na escolha das músicas para tocar na *EcoJukeBox*. Ao contrário dos filtros baseados em informação estática, os critérios dinâmicos requerem alguns cuidados, quer com a recolha da informação relevante necessária para os validar, quer com a atualização dessa

informação para o algoritmo se basear nos dados mais atuais possíveis. São estes critérios que tornam a escolha das músicas "mais inteligente", dado que têm em consideração todo o tipo de interações que acontecem no sistema, desde a adição de uma música, uma música que tocou ou uma interação de um utilizador com o sistema por intermédio de uma votação.

Os critérios dinâmicos que foram implementados neste projeto são os seguintes:

- Nunca ter sido tocada;
- Número de "não gostos";
- Número de "gostos";
- Número de votos num álbum;
- Número de votos numa anotação;
- Tocada recentemente;
- Não ter sido tocada recentemente;
- Adicionada recentemente;
- Não ter sido adicionada recentemente.

### 3.4.4 Peso dos critérios

Para cada critério a incluir no perfil de grupo é possível definir um peso, que vai fazer com que os critérios não tenham a mesma relevância na pontuação a atribuir a cada música. Estes pesos só são aplicados às preferências, pois relativamente às exigências apenas importa saber se a música cumpre ou não esses requisitos, não fazendo sentido ter um valor numérico associado. Esses valores são definidos aquando da criação de um perfil de grupo. Nessa altura são definidas as exigências e preferências do grupo e a cada critério será definido o valor a ponderar sobre a música.

## 3.5 Eventos

Durante a reprodução de músicas num espaço musical comum, ocorrem vários eventos que são necessários identificar e em alguns casos obter e guardar informações sobre os mesmos. Neste projeto identificam-se vários eventos, tais como, adicionar uma nova música, votação com "gosto", votação com "não gosto", nova anotação associada a uma música, votação num álbum, votação numa anotação e música tocada. Foi necessário guardar informação de que esses eventos ocorreram para depois determinar a sua influência na classificação das músicas. A tabela seguinte descreve os diferentes eventos existentes na *EcoJukeBox*.

| Eventos | Descrição |
|---|---|
| Adicionar uma nova música | Este evento ocorre quando é adicionada uma nova música na *EcoJukeBox.* |
| Votação com "gosto" | Este evento ocorre quando um utilizador vota com "gosto" na música que está a tocar atualmente. |
| Votação com "não gosto" | Este evento ocorre quando um utilizador vota com "não gosto" na música que está a tocar atualmente. |
| Nova anotação associada a uma música | Este evento ocorre quando um utilizador associa uma anotação à música que está a tocar atualmente. |
| Votação num álbum | Este evento ocorre quando um utilizador vota num determinado álbum. |
| Votação numa anotação | Este evento ocorre quando um utilizador vota numa determinada anotação. |
| Música Tocada | Este evento ocorre quando é escolhida para tocar uma nova música na *EcoJukeBox.* |

**Tabela 3-1: Descrição dos eventos da *EcoJukeBox***

## 3.6 Seleção da música a tocar

A seleção das músicas a tocar está totalmente ligada à validação e classificação das mesmas, de acordo com o perfil do grupo. A música escolhida para tocar será sempre aquela que, tendo satisfeito todas as exigências, foi a mais bem classificada, sendo que em caso de empate escolhe-se uma das mais bem classificadas ao acaso. A classificação das músicas vai sendo alterada ao longo do tempo de reprodução de músicas na *EcoJukeBox*. O sistema sabe que critérios estão a ser usados por aquele espaço musical e cada vez que uma música é escolhida para tocar recalcula as classificações das músicas, fazendo com que a seleção musical se baseie na informação o mais atualizada possível. Existem alguns critérios que são facilmente calculados, principalmente os que são classificados como critérios baseados em informação estática, pois apenas é necessário verificar se a música cumpre ou não o critério. No entanto, para os critérios baseados em informação dinâmica foi necessário aplicar uma fórmula de normalização típica, que conseguisse distribuir equitativamente as classificações de cada música, tendo em conta o valor atual e o menor valor. Existem duas fórmulas de normalizar as músicas, pela positiva ou pela negativa.

### 3.6.1 Seleção de música pela positiva

Os critérios baseados em informação dinâmica que têm como objetivo selecionar uma música pela positiva são: *música tocada recentemente*, *música adicionada recentemente*, *número de "gostos"*, *número de votos num álbum* e *número de votos numa anotação*. Para tal, foi usada uma fórmula que atribui valores entre zero e um, conforme o valor da música se aproxime ou não do valor máximo que pode atingir.

Por exemplo, para determinar se uma *música foi tocada recentemente* é necessário ter em conta três fatores: o tempo atual da emissão, o tempo em que a música tocou pela última vez e o tempo da música que tocou há mais tempo.

Para calcular o valor a atribuir a uma música, usando o critério *música tocada recentemente*, usa-se a seguinte fórmula:

$$V = 1 - \frac{Nemissao - Vm}{Nemissao - MenorValor}$$

De acordo com a fórmula, subtrai-se o instante em que a música tocou (Vm) ao valor atual da emissão (Nemissao) e divide-se pela diferença entre o valor atual e o menor valor (MenorValor). Esse valor subtraído a um dá-nos o novo valor a atribuir à música.

Neste projeto, cada música tocada corresponde a uma unidade de tempo. Caso a *EcoJukeBox* já esteja a tocar há quinze unidades de tempo, e querendo aplicar o critério das músicas mais recentes, vamos ver que valores tomam duas músicas M1 e M2. A música M1 tocou a última vez há catorze unidades de tempos e a música M2 tocou a última vez há duas unidades de tempo.

$$Vm1 = 1 - \frac{15 - 14}{15 - 0} = 0{,}94(4)$$

$$Vm1 = 1 - \frac{15 - 2}{15 - 0} = 0{,}14(4)$$

Como se pode observar, a música M1, por ter tocado mais recentemente, tem um valor maior do que a música M2 que já tocou há algum tempo. Logo a música M1 teria maior pontuação do que a música M2.

Para os outros critérios enumerados acima, poderão ser consultadas as respetivas fórmulas no Apêndice H.

### 3.6.2 Seleção de música pela negativa

Os critérios baseados em informação dinâmica que têm como objetivo selecionar uma música pela negativa são: *música não ter sido tocada recentemente*, *música não ter sido adicionada recentemente* e *número de "não gostos"*. Nos critérios pela negativa, usa-se uma fórmula inversa da anterior. É usada em alguns dos critérios deste projeto, como por exemplo, para saber se uma *música não foi tocada recentemente*, ou seja, dando prioridade às músicas que já não tocam há mais tempo e com isso não repetindo tantas vezes as músicas.

Tal como no caso anterior, também é necessário ter três fatores em consideração: o tempo atual da emissão, o tempo em que a música tocou pela última vez e o tempo da música que tocou há mais tempo.

Para calcular o valor a atribuir a uma música, usando o critério *música não foi tocada recentemente*, usa-se a seguinte fórmula:

$$V = \frac{Nemissao - Vm}{Nemissao - MenorValor}$$

Usando o mesmo exemplo das duas músicas M1 e M2 que foram descritos acima, obtêm-se os seguintes resultados:

$$Vm1 = \frac{15 - 14}{15 - 0} = 0{,}06(6)$$

$$Vm2 = \frac{15 - 2}{15 - 0} = 0{,}86(6)$$

Neste caso, verifica-se que a música M1 tem um peso menor por ter tocado mais recentemente do que a música M2.

Relativamente aos restantes critérios enumerados anteriormente, poderão também ser consultadas as respetivas fórmulas no Apêndice H.

Podem ainda existir critérios baseados em informação dinâmica que não precisam de fórmulas para serem calculados, como é o caso do critério *música nunca ter sido tocada*, uma vez que é necessário apenas fazer uma verificação da sua veracidade.

## 3.7 Atualização das pontuações das músicas

Cada vez que uma música é escolhida para tocar, a que tiver melhor pontuação no momento, é necessário atualizar as pontuações das músicas. Começa-se por verificar para todas as músicas se estão ilegíveis para tocar, ou seja, se não cumprem todas as exigências. Para as restantes músicas, as que cumprem as exigências, recalcula-se as suas pontuações. Nesse processo procede-se à soma dos valores obtidos pelos critérios, de acordo com as fórmulas descritas acima nos pontos 3.6.1 e 3.6.2 multiplicados pelos pesos definidos para cada critério aquando da criação do perfil de grupo.

# Capítulo 4

# Desenho

## 4.1 Arquitetura

A arquitetura física do sistema assenta no modelo cliente-servidor (Figura 4.1). Cada instância de um cliente pode enviar pedidos de serviços que serão fornecidos por uma ou mais máquinas.

O servidor da aplicação escolhido foi o *Glassfish* [24] [glossário: G]. É um servidor de código fonte aberto da *Oracle* usado na plataforma *Java Enterprise Edition* [25] [glossário: G]. O *Glassfish* suporta todas as especificações da *API Java EE*, tais como *JPA* e define como coordená-las. Também suporta diversas tecnologias de serviços na *Web*. Outra das razões desta escolha foi a grande quantidade de informação disponível na *Web*, que ajudou bastante na configuração destes servidores. Este servidor é o responsável por alojar a componente servidor da aplicação e fornecer os serviços na *Web SOAP* necessários para comunicar quer com o *Player*, quer com a aplicação *Web*. É responsável também por comunicar com a base de dados *MySQL*, onde vai ser persistida a informação. O servidor *Glassfish* está disponível no porto 8080.

O *Player*, desenvolvido em *C#* [26] [glossário: G] é responsável por fazer o *streaming* das músicas para um servidor de rádio, *Icecast 2* [23] [glossário: G]. Este servidor está disponível no porto 8000. O *Player* comunica com o servidor de *Glassfish* para obter a informação da música que vai tocar.

A interação com o utilizador é feita através de um navegador. São enviados pedidos aos serviços na *Web* conforme as ações efetuadas pelos utilizadores. Em sentido contrário, são enviados também via serviços na *Web* para um servidor *apache*, no porto 1234, as informações a apresentar no navegador sobre músicas, álbuns e anotações. Os dados que circulam na rede e provenientes do servidor da aplicação são serializados e são enviados pela rede em formato *JSON* [27] [glossário: G].

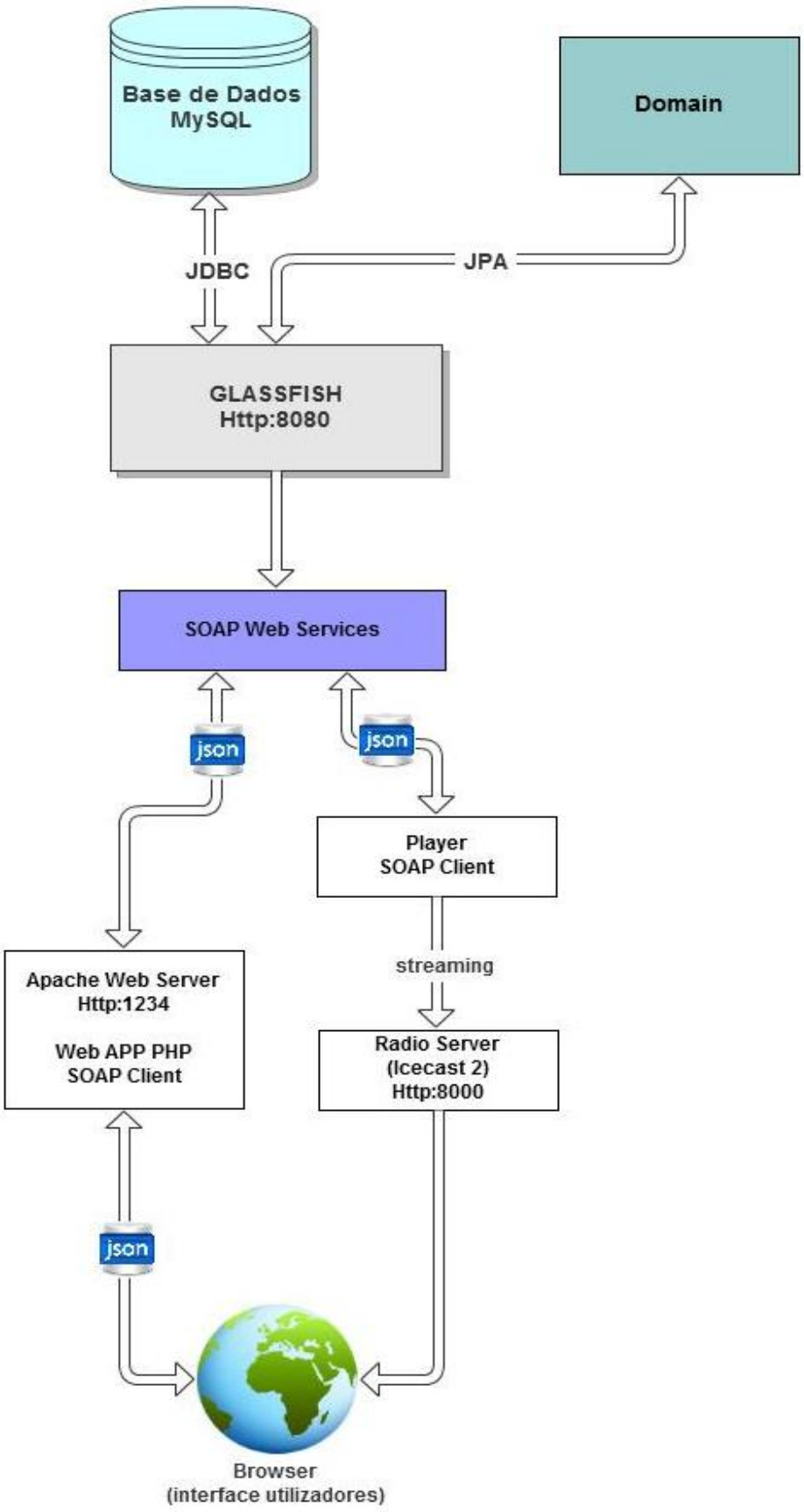



**Figura 4-1: Arquitetura da *EcoJukeBox***

## 4.2 Conteúdo musical

A reprodução de músicas recorre essencialmente a uma fonte musical interna, onde estão armazenadas as músicas que serão possíveis reproduzir na *EcoJukebox*. É ainda usada uma fonte musical externa, a *Last.Fm* [11] para obter informações complementares necessárias sobre cada música.

### 4.2.1 Fonte musical interna

A fonte musical interna é responsável por guardar as músicas que podem ser tocadas na *EcoJukeBox*. O repositório de música da *EcoJukeBox* poderá sofrer alterações, pois qualquer computador ou dispositivo móvel pode passar a sua própria música para ser partilhada e tocada. O esquema abaixo apresentado ilustra essa interação.

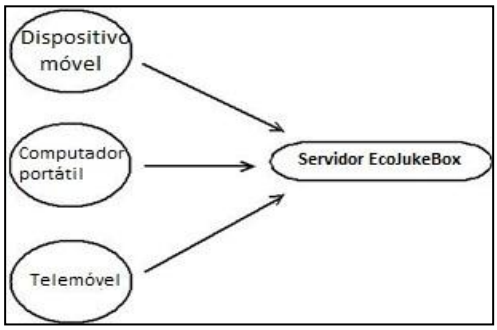



**Figura 4-2: Suporte de músicas da *EcoJukeBox***

A tabela seguinte apresenta uma comparação entre alternativas fáceis de partilhar informação [12]:

| | **Serviços para armazenamento e sincronização de ficheiros online** | | | | |
|---|---|---|---|---|---|
| | **Dropbox** [28] | **Sky Drive**[29] | **SpiderOak** [30] | **SugarSync** [20] | **Wuala** [32] |
| **Sistemas Operativos Suportados** | Windows, Mac, Linux | Windows, Mac | Windows, Mac, Linux | Windows, Mac | Windows, Mac, Linux |
| **Sistemas Móveis Suportados** | iOS, Android, BlackBerry | N/A | iOS, Android, Maemo | iOS, Android, BlackBerry, Windows Mobile | iOS, Android |
| **Capacidade de Armazenamento** | 2 Gb | 5 Gb | 2 Gb | 5 Gb | 1 Gb+ |

**Tabela 4-1: Comparação entre *software* de sincronização de ficheiros**

Atendendo às características apresentadas pelos vários serviços para armazenamento e sincronização de ficheiros *online*, aquele que tem uma maior aplicabilidade no projeto em causa é o *SugarSync*, pois permite uma maior capacidade de armazenamento e também uma maior gama de telemóveis suportados.

### 4.2.2 ID3 Tags

#### Ler informações de ficheiros *MP3*

Na leitura de ficheiros de música do tipo *MP3*, é extremamente útil ter acesso à informação sobre essa música, como o artista, o título da música, título do álbum, etc. Esta meta informação é guardada sob a forma de *ID3 Tags* [13], que a armazena, no arquivo, de forma estruturada.

Neste momento as *Tags ID3* são lidas por *software* como o *iTunes* [32], o *Windows Media Player* [33], o *Winamp* [34], o *MusicMatch* [35] e dispositivos como o *iPod*, *Creative Zen*, *Toshiba Gigabeat* e *Sony Walkman*. Este tipo de anotações apenas pode ser usado em ficheiros do tipo *MP3*. Para outros formatos existem outros tipos de anotações. Os ficheiros do tipo *WMA* e *AAC* têm o seu próprio formato de anotações, os ficheiros *Ogg Vorbis* usam "*Xiph Comments*" e os ficheiros *WAV* não têm anotações.

Ao contrário da primeira versão do *ID3* que apenas permitia 128 *bytes* de dados, adicionados ao final de cada ficheiro *MP3*, a atual versão de *ID3 tags* (ID3 v2) [36] tem tamanho variável, e usualmente ocorre no início de cada ficheiro para ajudar no *streaming* de média. Consiste num conjunto de *frames*, cada um dos quais contendo pedaços de meta-dados. Por exemplo, a *frame TIT2* contém o título da música e a *frame WOAR* contém o *URL* do site do artista. As *frames* podem ser de até 16Mb de comprimento, enquanto que o tamanho total da anotação é limitada a 256Mb.

Existem três versões do ID3 v2:

- ID3v2.2 foi a primeira versão pública do ID3v2, lançada em 1998. Este *standard* é considerado obsoleto.
- ID3v2.3 expandiu o identificador de *frame* para quatro caracteres e acrescentou uma série de *frames*. Uma *frame* pode conter vários valores, separados por um caracter. Esta é a versão mais utilizada de ID3v2.

- ID3v2.4 é a última versão publicada. Permite que os dados textuais sejam codificados em UTF-8 em vez de UTF-16. Permite também a adição de uma anotação para o final do arquivo antes de outras anotações (como ID3v1).

Neste trabalho é usado a versão ID3v2.3, por ser a mais usada, permitindo assim um acesso a mais documentação e exemplos ilustrativos do seu funcionamento. Por um lado no trabalho tenta-se primeiro obter informação, caso exista, sobre a música através das *Tags ID3*. Posteriormente essa informação é complementada usando a *Last.Fm*, como iremos ver de seguida. Depois de obtido o máximo de informação sobre a música na *Last.Fm*, atualizamos as *ID3 Tags* do ficheiro *MP3*.

### 4.2.3 Fonte musical externa

A informação contida em *ID3 Tags* é escassa, por isso é necessário recorrer à *API* da *Last.Fm* [11], que disponibiliza uma vasta gama de serviços, permitindo obter conhecimento sobre conteúdos discográficos, o que possibilita saber alguns dados complementares acerca de cada música, tais como: o nome do álbum, a duração da música e as anotações.

Para aceder aos serviços disponibilizados é necessário uma chave de acesso à *API* da *Last.Fm*. Esta chave identifica o utilizador, evitando usos abusivos. A *API* da *Last.Fm* está dividida em 15 categorias, no entanto, apenas serão analisados os serviços utilizados no projeto, apresentando a sua descrição e função.

**Categoria Música:**

- track.getInfo(): Obtem os meta-dados sobre uma determinada música
- track.getName(): Obter o nome da música
- track.getAlbum(): Obter o álbum de uma determinada música
- track.getArtist(): Obter o autor da música
- track.getDuration(): Obter a duração da música
- getImage(track): Obter a capa do álbum

**Categoria Anotação:**

- tag.getTopTags(): Obter as 40 anotações de topo existentes na *Last.Fm*, classificadas por popularidade (ordenado por número de vezes usadas).

## 4.3 Persistência (JPA)

A Persistência é uma camada de alto nível relacional do *JDBC*. A camada de persistência mapeia objetos para uma base de dados.

A persistência em *Java EE* tem uma especificação própria, chamada *Java Persistence API* [14]. Esta *API* define a forma regular de mapear *POJOs (Plain Old Java Object)* [glossário: G] numa base de dados. *POJO* é o termo utilizado para definir objetos simples. Esses objetos *Java* comuns são chamados de entidades. As entidades *Java*, são mapeadas para uma base de dados utilizando anotações *JPA*. Portanto estes objetos podem ser inseridos e carregados de uma base de dados sem que seja necessário escrever nenhum código de conexão *JDBC* [glossário: G]. O *JPA* também define uma linguagem de consulta (*JPQL*) que tem características semelhantes ao *SQL*, mas é adaptada para trabalhar com objetos *Java*.

### 4.3.1 Classe Entidade

A *entity class* é uma entidade que representa o objeto que se quer guardar na base de dados. Deve ter uma anotação *@Entity* e, pelo menos, um campo ou método *getter* que é designado como a chave primária da classe. Isso deve ser feito com a anotação *@Id*. As entidades têm disponíveis outro tipo de anotações para definir o esquema relacional entre elas, para que essas relações possam também ser mapeadas para a base de dados.

### 4.3.2 Gestor de Entidades

As entidades são agrupadas num conjunto finito de classes chamado de unidade de persistência. Existe uma classe *Entity Manager* que gere um *persistence unit*. Cada unidade de persistência deve estar associada a uma base de dados específica, para que o gestor de entidades possa saber onde, como e com que tipo de base de dados está a interagir. Esta informação é guardada num ficheiro *XML*, chamado *persistence.xml*.

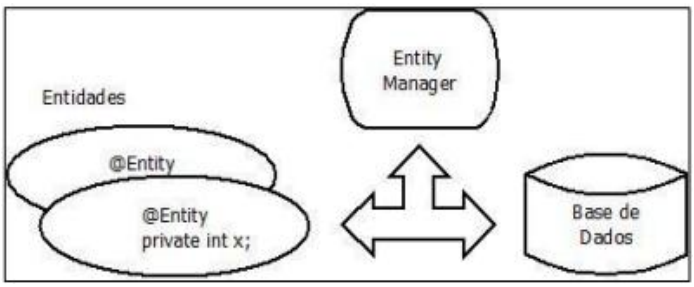



**Figura 4-3: Arquitetura *JPA***

A *Java Persistence API* (*JPA*) foi usada neste trabalho como implementação do padrão *Data Mapper* [22], que apresenta uma solução para o mapeamento objeto-relacional entre o modelo de domínio orientado a objetos utilizado na aplicação (*Java EE*), e a base de dados relacional subjacente.

### 4.3.3 Relações

Existem quatro tipos de cardinalidade: um-para-um, um-para-muitos, muitos-para-um e muitos-para-muitos. Além disso cada relação pode ser unidirecional ou bidirecional. As anotações para este tipo de relações são *@OneToOne*, *@OneToMany*, *@ManyToOne* e *@ManyToMany*.

Importa também falar sobre a anotação: *cascade()*. Esta anotação indica que quando uma operação é executada sobre uma entidade, esta operação também seja realizada em todos os relacionamentos que essa entidade tem.

### 4.3.4 Operações

Existem quatro operações que podem ser realizadas através do *Entity Manager*: *Persist*, *Merge*, *Remove* e *Refresh*. *Persist* é usado para criar entidades na base de dados. O *Merge* lida com a sincronização de uma entidade, com inserções e atualizações. O *Remove*, tal como o nome indica é usado para remover entidades da base de dados. O *Refresh* é semelhante ao *Merge*. Ele não atualiza a base de dados com as mudanças na instância do objeto. Em vez disso, ele atualiza o estado da instância do objeto da base de dados.

### 4.3.5 Consultas

A *API* do *Java Persistence* fornece uma linguagem para efetuar consultas à base de dados e poder referenciá-las pelo seu nome, aquando da criação de uma interrogação à base de dados. Este mecanismo tem o nome de *Named Queries* e tem como vantagem poder reutilizar as mesmas consultas à base de dados em várias situações diferentes. Este mecanismo tem como anotação *@NamedQuery*.

Podemos dividir a base de dados em dois grupos distintos: música e perfis de grupo. Relativamente à informação sobre as músicas existem quatro entidades associadas entre si: *music*, *poll*, *album* e *tag*. Sobre os perfis de grupo existem as entidades *profile*, *criteria*, *preferences* e *requirements*.

Em relação às músicas é importante não só ter a informação das anotações associadas a cada música, mas também saber as músicas a que correspondem determinada etiqueta. O mesmo sucede para o caso dos álbuns. É importante saber a que álbum corresponde cada música, mas também saber em relação a cada álbum as músicas que lhe estão associadas. Cada música tem também associada uma votação, onde registamos o número de gostos e não-gostos que cada música recebeu.

As músicas podem ter várias anotações associadas e as anotações podem estar associadas a várias músicas (*@ManyToMany*). Os álbuns pertencem a diferentes músicas (*@OneToMany*), mas uma música tem apenas um álbum (*@ManyToOne*). Cada votação está associada a apenas uma música e cada música tem apenas uma votação (*@OneToOne*). Todas estas relações são bidirecionais.

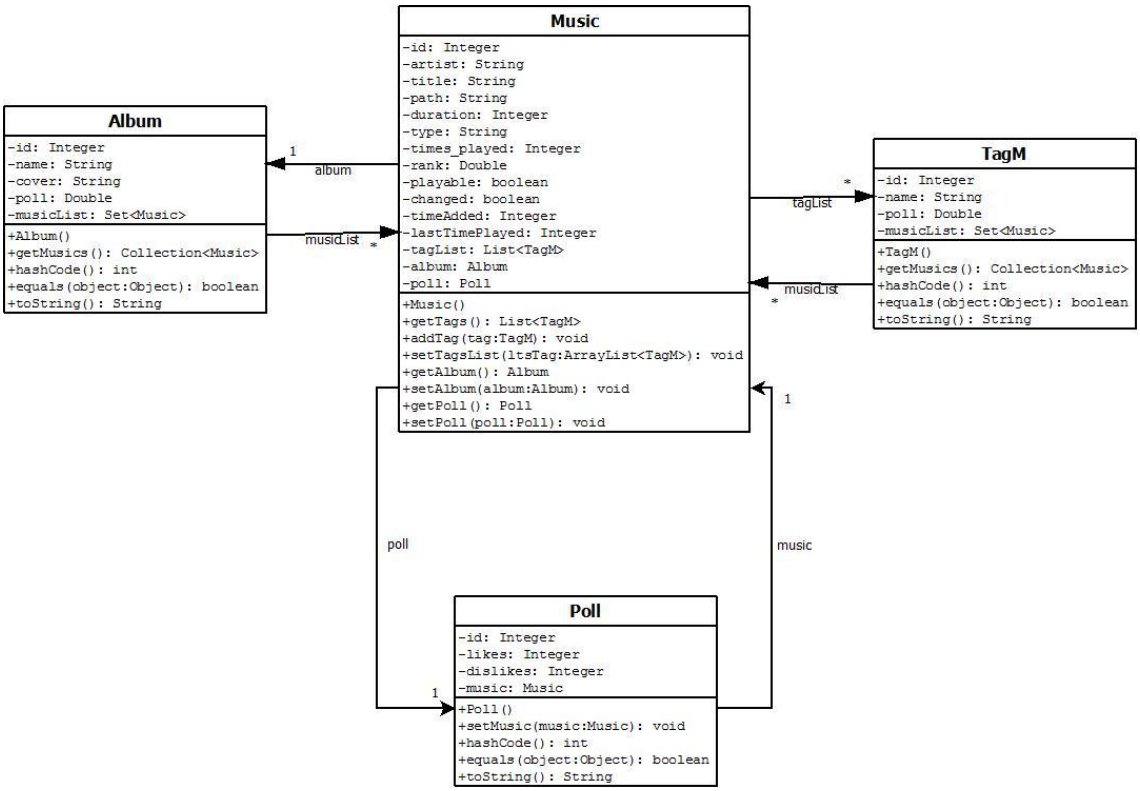



**Figura 4-4: Diagrama de classes relativo às músicas**

Em relação aos perfis de grupo, têm associados a si, preferências e exigências. Às exigências e preferências estão associados uma série de critérios. Cada critério pode ser uma exigência, uma preferência, ou até ambas.

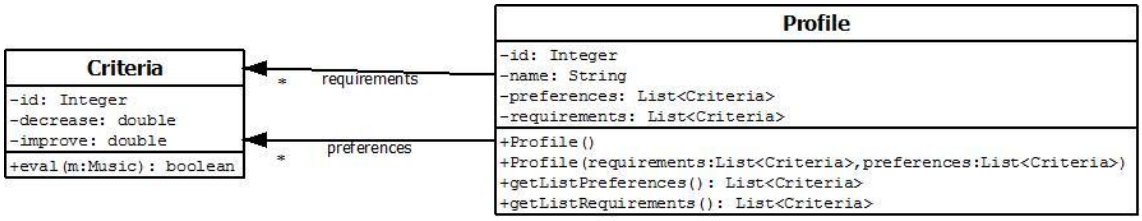



**Figura 4-5: Diagrama de classes relativo aos perfis de grupo**

## 4.4 Padrões de Desenho Utilizados

### 4.4.1 Padrão *Command*

O padrão *command* [15] é uma forma de encapsular a invocação de um método. Dá-nos a capacidade de parametrizar e passar em torno de uma chamada a um método, que pode ser executado sempre que seja necessário. Todos os objetos de comando dentro do código vão implementar os mesmos métodos (que são usualmente chamados *execute*, *run*, ou *undo*). Através do uso de interfaces, é possível criar classes que podem executar estes *commands* sem precisar de saber nada sobre eles, além de garantirem uma correta implementação da interface.

#### O Cliente, o Invocador e o Recetor

Existem três atores neste sistema: o cliente, o objeto invocador e o objeto recetor. O cliente instancia o comando e passa-o ao invocador. O invocador por sua vez recebe o comando e guarda-o. A certa altura, pode chamar o método execute do *command*, ou pode passar o *command* para outro invocador potencial. O recetor é o objeto que realiza realmente a ação.

Em suma: o cliente cria o *command*. O invocador executa o *command*. O recetor executa a ação quando o *command* é executado. Com a exceção do cliente, os nomes são um pouco descritivos do que fazem.

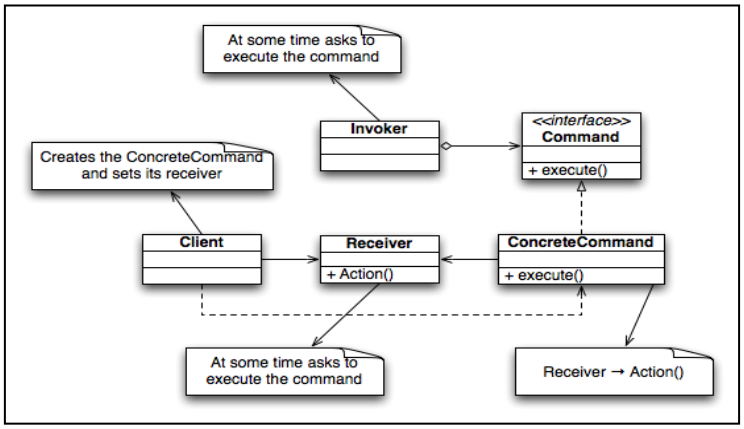



**Figura 4-6: Diagrama de classes do Padrão *Command***

### Uso do Padrão *Command*

O principal objetivo do padrão *command* é dissociar o objeto invocador (*UI*, *API*, *proxy*, etc) do objeto a executar a ação. Foi utilizado para garantir uma maior modularidade na interação entre dois objetos. Com a ajuda deste padrão foi possível normalizar ações de modo a que uma única classe de invocador pudesse chamar uma grande variedade de métodos, sem saber nada sobre eles.

O padrão *command* quando utilizado corretamente torna o código mais modular e flexível. Neste projeto foi utilizado para executar ações sobre os diferentes eventos que ocorrem no sistema, permitindo assim que para cada evento se defina a ação adequada.

## 4.4.2 Padrão *Singleton*

O *Singleton* [16] é um dos padrões mais básicos, mas também dos mais usados. É uma maneira de fornecer acesso global a uma instância de classe, sem disponibilizar o construtor fora da classe. A classe *Singleton* instancia-se e mantém essa instância disponível para múltiplos processos. O principal benefício do padrão *Singleton* é a maneira como ele organiza o seu código. Ao agrupar os métodos relacionados e atributos juntos num único local, que não pode ser instanciado várias vezes, é mais fácil de corrigir e mantê-lo. Permite também um controlo sobre como e quando as instâncias são acedidas. Um outro benefício deste padrão é uma maior flexibilidade que os métodos estáticos, pois permite polimorfismo.

Neste trabalho existem vários exemplos de utilização do padrão *Singleton*. Num dos casos este padrão de desenho é usado para garantir que o *EntityManager-Factory* seja único. Sendo este o ponto de acesso à base de dados é importante ter em conta que cada vez que fosse necessário fazer uma chamada à base de dados, não fosse criada uma nova instância do objeto. Assim existe um ponto único de acesso à base de dados ao qual se pode aceder em toda a aplicação. Outro dos casos de *Singletons* no projeto são os controladores. Também é importante manter apenas uma instância de cada controlador, pois são eles que gerem toda a informação e todos os processos que fazem parte da *EcoJukeBox*.

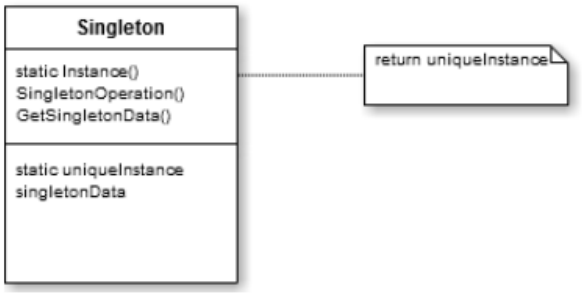



**Figura 4-7: Diagrama de classes do padrão *Singleton***

### 4.4.3 Padrão *Facade*

O padrão *facade* [16] tem dois propósitos: simplificar a interface de um sistema e separar essa classe do código do cliente que a utiliza. Tenta simplificar a interface de um sistema na medida em que pode incorporar várias bibliotecas numa classe e disponibiliza métodos que sejam convenientes para o uso de sistemas mais complexos. Este padrão simplifica tarefas comuns ou repetitivas, tais como registos de erros ou manter controlo de interações com o sistema, tal como é usado neste projeto. Propõe também combinar vários métodos num só, caracterizando-os de forma a que sejam mais facilmente utilizados. Os controladores usados neste projeto são um bom exemplo da aplicabilidade deste padrão. Cada controlador disponibiliza uma série de métodos que vão ser utilizados pelos clientes e cada classe controlador usa uma série de bibliotecas e outros métodos.

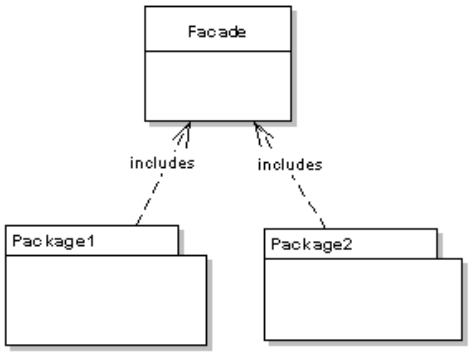



**Figura 4-8: Diagrama de classes do Padrão *Facade***

### 4.4.4 Padrão *Strategy*

O padrão *strategy* [17] define uma família de algoritmos, encapsula cada um deles, e permite que sejam usados alternadamente com o mesmo propósito. Estratégias são úteis quando é necessário decidir em tempo de execução qual o algoritmo a executar em cada caso. Podem ser utilizados para variar o uso de cada algoritmo de acordo com o contexto.

Este padrão *strategy* tem três componentes principais: a estratégia, a estratégia concreta, e o contexto. A estratégia é uma interface comum que une todos os algoritmos suportados. Normalmente, essa classe é abstrata. A estratégia concreta é a classe de implementação, que representa cada algoritmo do grupo. O contexto é o objeto que une uma estratégia concreta a um fluxo de código particular.

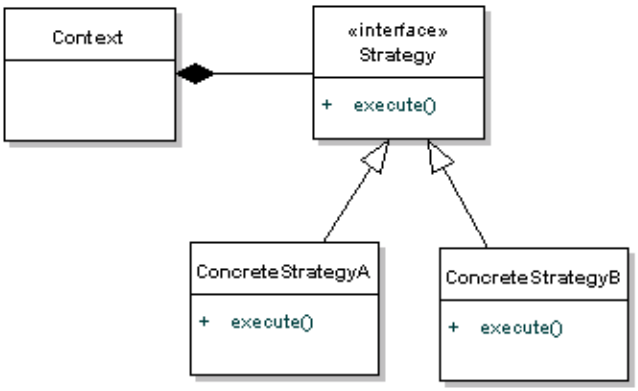



**Figura 4-9: Diagrama de classes do Padrão *Strategy***

Neste projeto o padrão *strategy* é usado nos critérios de seleção de música. Cada critério tem o seu comportamento específico, mas todos os critérios têm em comum o facto de influenciarem a escolha das músicas e terem associados a si um peso. Como neste caso temos uma família de classes com uma estrutura igual, mas que diferem no seu comportamento, o padrão *strategy* aplica-se a este problema em concreto.

### 4.4.5 Padrão DAO (*Data Access Object*)

Um erro típico de desenho que os programadores inexperientes cometem é misturar diferentes tipos de lógica (por exemplo, lógica de apresentação, lógica de negócio e lógica de acesso a dados) num único módulo. Isto reduz a reutilização do módulo e facilidade de manutenção, devido ao forte acoplamento. O objetivo geral do padrão *Data Access Object (DAO)* [18] é o de evitar esses problemas, separando a lógica de acesso a dados da lógica de negócio e lógica de apresentação. Este padrão recomenda que a lógica de acesso aos dados seja encapsulada num módulo independente chamado *data access object*.

Desta forma, o padrão *DAO* propõe que todas as operações principais de acesso à base de dados (adicionar, remover, atualizar) numa classe genérica que seja transparente para o resto da aplicação (ver modelo de classes no Apêndice A). A vantagem desta classe

ser genérica é o facto de permitir tratar de forma idêntica as operações descritas acima nos vários objetos do sistema: *Tag*, *Profile*, *Poll*, *Music*, *Album*.

No caso de ser necessário implementar uma nova operação mais concreta em relação a um objeto específico existe a possibilidade de estender a classe. Neste trabalho foi necessário estender a classe *DAO<E>* para definir vários métodos que operam especificamente em relação a cada objeto sobre a base de dados. A maior parte desses métodos são pesquisas, por exemplo, para obter uma determinada música da base de dados dado o seu título ou um determinado *album*, *tag*, ou mesmo um conjunto de objetos que cumpram um determinado critério.

### 4.4.6 Padrão DAO *Factory*

O padrão *DAO* proporciona às aplicações um grau de independência de todas as operações de acesso à base de dados. Mas a aplicação precisa ainda de saber qual o *DAO* apropriado a instanciar e fornecer as informações de conexão à base de dados, tais como, endereço de servidor, *passwords*, nome do servidor, etc. Para os casos em que é necessário suportar conexões a diferentes bases de dados, deve ser utilizado o padrão *DAO Factory*, pois permite abstrair o processo de encontrar um mecanismo de persistência adequado longe dos componentes de apresentação/negócio. Assim as aplicações interagem apenas com uma única classe, que produz tantas classes *DAO* quantas forem necessárias. Cada classe *DAO* implementa uma interface específica, facilitando a mudança de mecanismos de persistência sem afetar a aplicação como um todo.

Neste trabalho é usada como *DAO Factory* a classe *DAOConnect* (ver modelo de classes no Apêndice A), responsável por criar o *Factory* de gestor de entidades *DAO* e por fazer a ligação com a base de dados.

### 4.4.7 Padrão Data *Mapper*

O padrão *Data Mapper* [22] sugere a criação de uma camada que torna independentes os objetos *Java* em relação à base de dados onde estes vão ser persistidos. Esta camada é constituída por vários objetos do tipo *Mapper*, que estabelecem a comunicação entre dois objetos independentes.

Este padrão é especialmente útil, pois objetos e bases de dados relacionais têm mecanismos diferentes para a estruturação dos dados. Por exemplo, a herança não é possível de representar numa base de dados relacional. Permite também que as

mudanças feitas na camada de domínio da aplicação não obriguem a uma reestruturação do esquema relacional da base de dados, devido à camada de abstração que os separa.

O *Data Mapper* é uma camada de *software* que separa os objetos das bases de dados, tendo a responsabilidade de fazer a ponte entre a aplicação e a base de dados e de os isolar. Com este padrão o esquema da base de dados é independente dos objetos que o utilizam e não tem qualquer informação sobre a camada de domínio da aplicação.

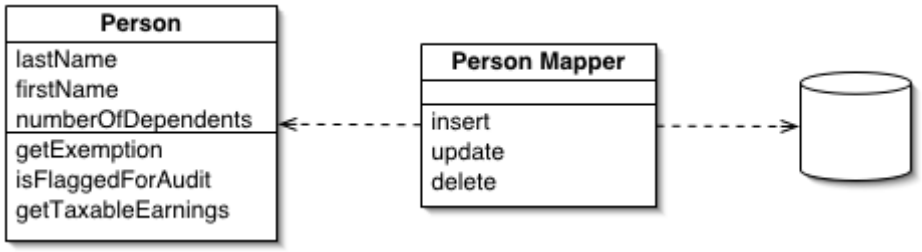



**Figura 4-10: Diagrama de classes do Padrão *Data Mapper***

## 4.5 Serviços na *Web*

### 4.5.1 Sobre os serviços na *Web*

Os serviços na *Web* [19] usam documentos *XML* criados no formato de mensagem, para o programa poder enviar um pedido para um serviço *Web* através da rede. As normas dos serviços na *Web* definem o formato da mensagem, especificando a interface para a qual a mensagem é enviada, descrevendo as convenções para mapear o conteúdo da mensagem para dentro e para fora dos programas que implementem o serviço, e definindo mecanismos para publicar e para descobrir interfaces de serviços na *Web*.

Esta tecnologia pode ser usada de várias formas. Os serviços na *Web* podem ser executados em clientes para aceder a aplicativos de *Internet*, tais como sistemas de reserva e sistemas de controlo de pedidos. Podem também ser usados para integração *B2B* (*business-to-business*), conectando aplicações executados em várias organizações na mesma cadeia de abastecimento. Os serviços na *Web* também podem facilitar a integração de aplicações empresariais (*EAI*), que significa a integração das aplicações de uma empresa, ou seja, interoperabilidade entre a informação que circula numa organização nas diferentes aplicações como, por exemplo, o comércio eletrónico com os seus clientes e os seus fornecedores. Esta interação constitui o sistema de informação de uma empresa. E para além da interoperabilidade entre as aplicações, a *EAI* permite definir um *workflow* entre as aplicações e pode constituir uma alternativa aos *ERP* (*Enterprise Resource Planning*). Com um *workflow* é possível otimizar e controlar processos e tarefas de uma determinada organização.

Como ilustrado na figura seguinte, os serviços na *Web* apresentam à rede uma maneira padrão de interface entre sistemas de software *back-end*, tais como sistemas de gestão de bases de dados, *.NET*, *J2EE*, *CORBA*, entre outros. As interfaces dos serviços na *Web* recebem uma mensagem *XML* padrão através da rede, transformando depois os dados *XML* num formato que seja entendido por um sistema de *software* específico e opcionalmente retorna uma mensagem de resposta. As implementações subjacentes aos serviços na *Web* podem ser criadas usando qualquer linguagem de programação, sistema operativo, ou sistema de *middleware*.

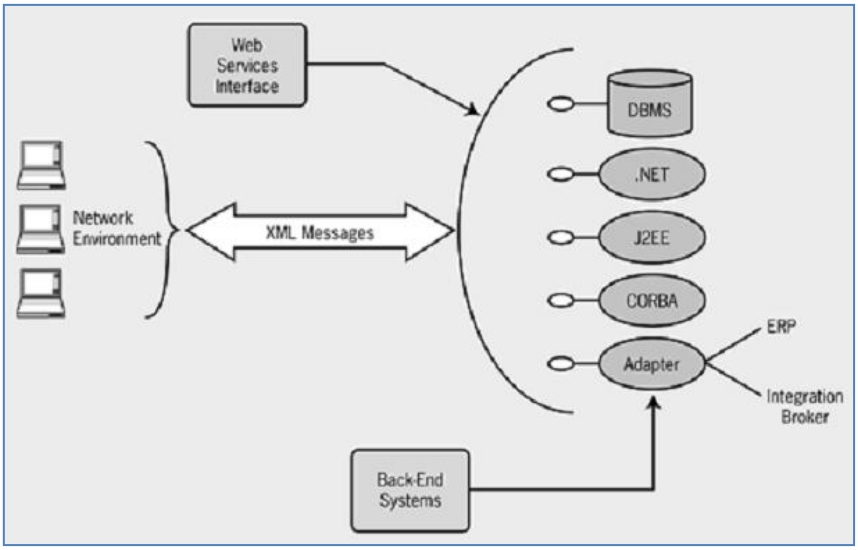



**Figura 4-11: Interface dos serviços na *Web* com os diferentes sistemas de *back-end***

Os serviços na *Web* combinam conceitos de programação com as características de abstração da *Internet*. As tecnologias da *Internet* são hoje um sucesso porque permitem um nível elevado de abstração e a compatibilidade entre sistemas operativos, *hardware* ou *software*. A infraestrutura dos serviços na *Web* explora esse nível de abstração e inclui informação semântica associada aos dados.

Os serviços na *Web* são importantes neste trabalho, pois permitem enviar as informações musicais através da rede até à interface *Web*, e a informação de retorno das votações na interface *Web* para o servidor da aplicação. Permite também a comunicação entre o servidor da aplicação e o servidor de rádio e é ainda usado para fazer a gestão da base de dados.

### 4.5.2 Tecnologias

As bases para a construção de um serviço na *Web* são os padrões *XML*, *WSDL* e *SOAP*.

#### XML - *Extensible Markup Language*

*XML* é a base em que os serviços na *Web* são construídos. O *XML* fornece a descrição, o armazenamento, o formato da transmissão para trocar os dados através dos serviços na *Web* e também para criar tecnologias de serviços na *Web* para a troca dos dados. A sintaxe de *XML* usada nas tecnologias dos serviços na *Web* especifica como os dados são representados genericamente, define como e com que qualidades de serviço os

dados são transmitidos, pormenoriza como os serviços são publicados e descobertos. Os serviços na *Web* decodificam as várias partes de *XML* para interagir com as várias aplicações.

## WSDL - *Web Services Description Language*

É um padrão baseado em *XML* que fornece o mecanismo pelo qual as definições dos serviços na *Web* são expostas e para manter conformidade nas mensagens *SOAP* enviadas. O *WSDL* descreve os tipos de dados e estruturas dos serviços da *Web*, explica como mapear os tipos de dados e estruturas para as mensagens que são trocadas, e inclui informações que ligam as mensagens subjacentes às implementações. É definido de modo a que as suas peças possam ser desenvolvidas separadamente e combinadas para criar um ficheiro *WSDL* abrangente. Os tipos de dados e estruturas podem ser partilhados entre várias mensagens.

A fim de conseguir uma comunicação de serviços da *Web*, o *WSDL* mapeia-os em protocolos de comunicação e transportes. Ambas as partes numa interação de um serviço da *Web* partilham um ficheiro *WSDL* comum. O remetente utiliza o arquivo *WSDL* para gerar a mensagem no formato adequado e usar o protocolo de comunicação adequado. O recetor usa o arquivo *WSDL* para entender como receber e analisar a mensagem e como mapeá-la para o objeto ou programa subjacente.

## SOAP - *Simple Object Access Protocol*

O *SOAP* é uma invocação remota de um método e para tal necessita especificar o endereço do componente, o nome do método e os argumentos para esse método. Estes dados são formatados em *XML* com determinadas regras e enviados normalmente por *HTTP* para esse componente. Não define ou impõe qualquer semântica, quer seja o modelo de programação, quer seja a semântica específica da implementação. Este aspeto é extremamente importante, pois permite que quer o serviço, quer o cliente que invoca o serviço sejam aplicações desenvolvidas sobre diferentes linguagens de programação. Por esta razão, o *SOAP* tornou-se uma norma aceite para se utilizar nos serviços na *Web*, uma tecnologia construída com base em *XML* e *HTTP*. Desta forma, pretende-se garantir a interoperabilidade e intercomunicação entre diferentes sistemas, através da utilização da linguagem *XML* e do mecanismo de transporte *HTTP* ou outro como, por exemplo, *SMTP*. O *SOAP* permite que os documentos *XML* de envio e de receção sobre a *Web* suportem um protocolo comum de transferência de dados para uma

comunicação de rede eficaz, ou seja, o *SOAP* providencia o transporte de dados para os serviços na *Web*.

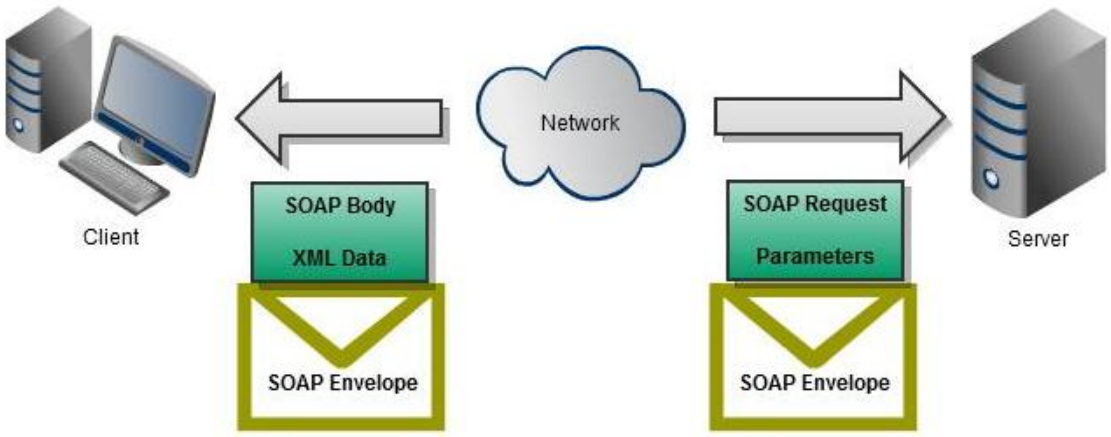



**Figura 4-12: Pedido e resposta *SOAP***

Todos os serviços *Web SOAP* usados no trabalho estão descritos no Apêndice B. Outro dos aspetos importantes em relação ao uso dos serviços na *Web*, são o tipo de dados devolvidos. No caso deste trabalho é necessário enviar muita informação pela rede, entre servidores, como é o caso das informações referentes a uma música. Tratam-se de objetos complexos que necessitam ser serializados para que os serviços consigam ser utilizados por outras linguagens de programação. Para serializar os objetos utiliza-se a estrutura de dados em *JSON*, que é um formato de texto padrão para troca de dados entre linguagens, tais como, *C*, *C++*, *C#*, *Java*, *JavaScript*, *Perl*, *Python*, e muitas outras.

Em *JSON* um objeto é uma *string*, representando um conjunto desordenado de pares nome/valor. Um objeto começa com uma chaveta curva esquerda ({) e termina com uma chaveta curva direita (}). Cada nome é seguido por dois pontos e os pares nome/valor são separados por vírgulas.

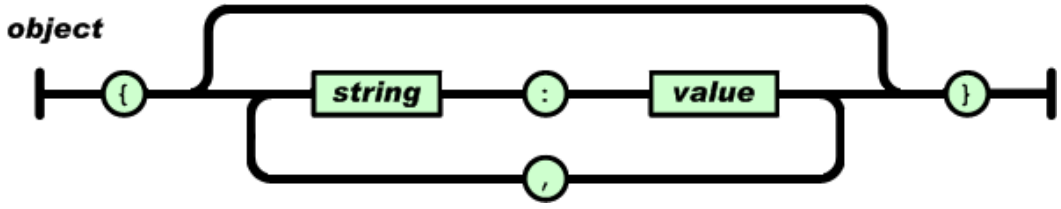



**Figura 4-13: Estrutura sintática de um objeto em *JSON***

A escolha desta estrutura de dados em *JSON* deveu-se a dois aspetos:

- Formato ideal para ser manipulado em *JavaScript* na interface *Web* do utilizador da *EcoJukeBox.*
- Existência de uma biblioteca feita em *Java* pela *Google* (*GSon*) que permite serializar objetos complexos para *JSON* de uma forma bastante simples de utilizar.

# Capítulo 5

# Avaliação da EcoJukeBox

Para efetuar testes criou-se um cenário em que a *EcoJukeBox* foi utilizada num espaço em que um grupo de pessoas partilha um espaço musical comum.

**Cenário do teste**

Este teste foi realizado num ambiente empresarial, num escritório, em que os vários elementos do grupo acederam à interface *Web* da aplicação *EcoJukeBox* através do seu portátil. Participaram cinco pessoas neste teste, com idades compreendidas entre vinte e cinco e trinta anos. Esta avaliação decorreu durante três dias, num total aproximado de sete horas e meia e com uma média diária de duas horas e meia.

Todos os participantes foram convidados a partilhar as suas coleções de música em formato .mp3. O estudo da avaliação consistiu em três partes: (1) uso do sistema, (2) um questionário efetuado após utilização do sistema e (3) uma discussão em grupo.

Como perfil de grupo foram definidos 5 preferências e nenhuma exigência, pois decidiu-se que não se queria excluir nenhuma música de tocar. As preferências foram discutidas entre os elementos do grupo, com os seguintes pesos:

- Número de gostos nas músicas (2)
- Número de gostos nos álbuns (2)
- Número de gostos nas anotações (8)
- Músicas adicionadas recentemente (10)
- Músicas não tocadas recentemente (12)

De acordo com estes critérios, o peso a atribuir na pontuação de uma música relativamente ao seu número de "gostos" varia entre 0 e 2. O mesmo sucede para cada

música em relação ao número de "gostos" no seu respetivo álbum. Uma música recebe também um peso entre 0 e 8, consoante o número de gostos nas anotações que lhe estejam associadas. Está também prevista um peso entre 0 e 10 para as músicas adicionadas recentemente, ou seja, as novidades que forem surgindo na *EcoJukeBox* têm uma bonificação para poderem tocar. Por outro lado as músicas que não tocaram recentemente têm também um peso entre 0 e 12, dando assim prioridade às músicas que já não tocam há mais tempo na *EcoJukeBox*.

No segundo dia de testes foi pedido pelos utilizadores para alterar o perfil do grupo, dando maior relevância às votações nos álbuns e aos "gostos" nas músicas. Foram alterados as seguintes preferências com novos pesos:

- Número de gostos nas músicas (5)
- Número de gostos nos álbuns (4)

Este pedido surgiu, pois os utilizadores queixavam-se que apesar de votarem com "gostos" nas músicas e nos seus álbuns, estas demoravam algum tempo a tocar ou não chegavam mesmo a tocar pois os pesos atribuídos a estes critérios eram baixos relativamente a outros critérios utilizados.

## 5.1 Uso do sistema

Durante os três dias de testes foram adicionadas 92 músicas na pasta partilhada. No primeiro dia foram adicionadas 58 músicas, no segundo dia 22 e no terceiro dia 12 músicas.

Foram votados 90 "gostos" nas músicas e 20 "não gostos". O número de "gostos" foi de 41 no primeiro dia, 25 no segundo e 24 no terceiro. Quanto aos "não gostos" foram 20, 0 e 6 respetivamente.

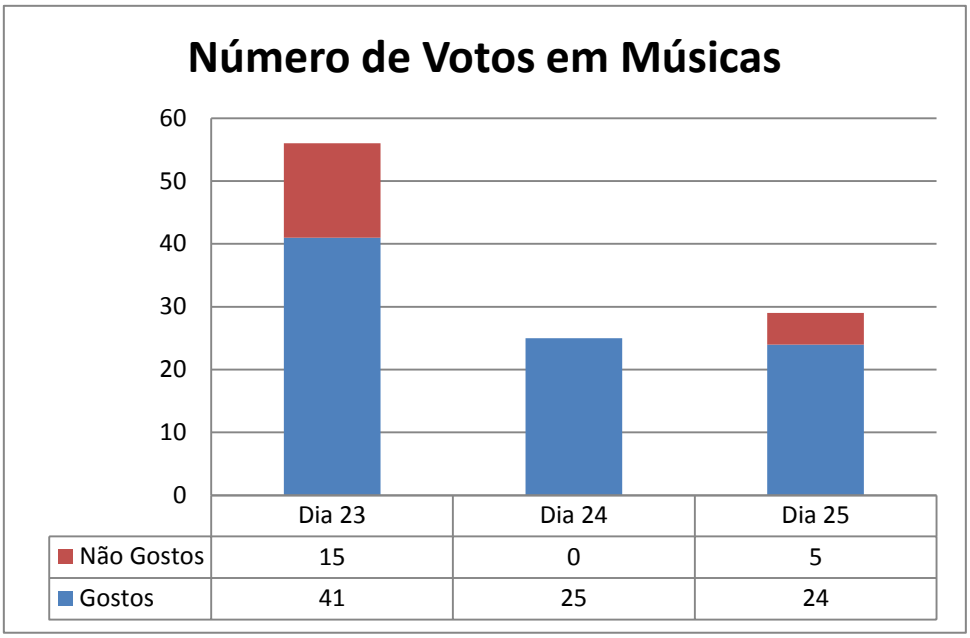



**Figura 5-1: Gráfico com o número de votos nas músicas tocadas na *EcoJukeBox***

Relativamente aos álbuns, foram contabilizadas 47 votações: 26 no primeiro dia, 10 no segundo e 11 no terceiro.

No que diz respeito a anotações, foram votadas 27 anotações ao longo dos 3 dias e adicionadas apenas 6. O número de votações foi de 12 no primeiro dia, 4 no segundo e 11 no terceiro. Foram adicionadas 5 anotações no primeiro dia e 1 no segundo dia.

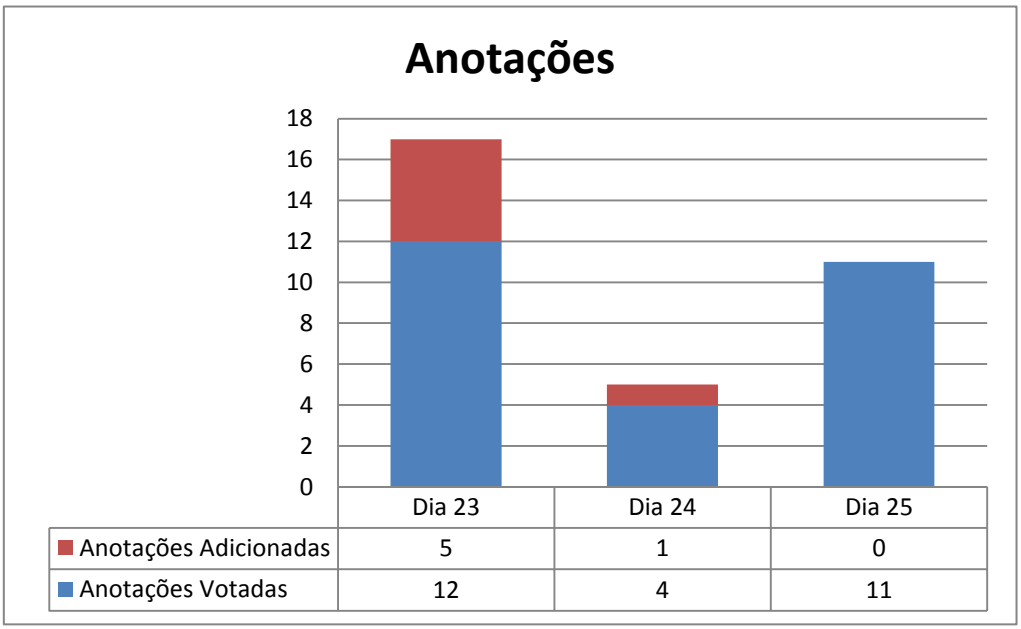



**Figura 5-2: Gráfico das anotações votadas e adicionadas na *EcoJukeBox***

Durante os três dias de testes foram tocadas 106 músicas, 48 no primeiro dia, 25 no segundo e 33 no terceiro.

Comparando os diferentes tipos de votações disponibilizadas ao utilizador para influenciar a escolha de músicas, pode-se verificar que as votações, do tipo "gosto" e "não gosto" foram as mais utilizadas. As votações em álbuns foram mais utilizadas comparativamente às votações em anotações. Já a adição de novas anotações teve muito pouca utilização por parte dos utilizadores. Isto pode querer dizer que o acesso à *Last.Fm* é suficiente, não havendo necessidade de um complemento de adição de anotações manuais.

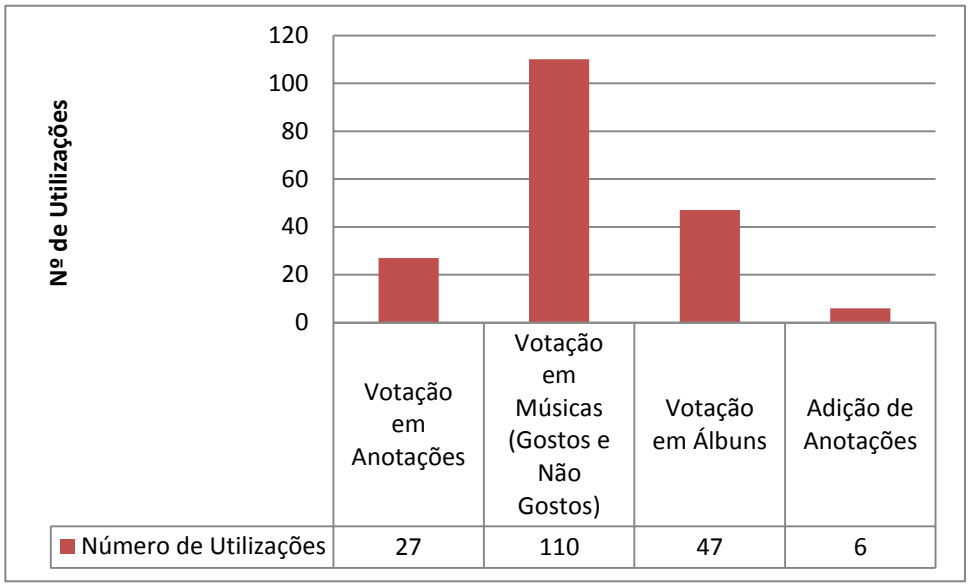



**Figura 5-3: Gráfico comparando as diferentes formas de interação na *EcoJukeBox***

A Figura 5-4 apresenta um gráfico com a distribuição do número de vezes que cada música tocou na *EcoJukeBox*. Como podemos verificar, a maioria das 92 músicas existentes na *EcoJukeBox* tocou pelo menos uma vez. Esta distribuição deveu-se ao facto de haver um critério no perfil delineado para este grupo que dava primazia às músicas adicionadas recentemente e um outro critério que dava prioridade às músicas que não eram tocadas há mais tempo.

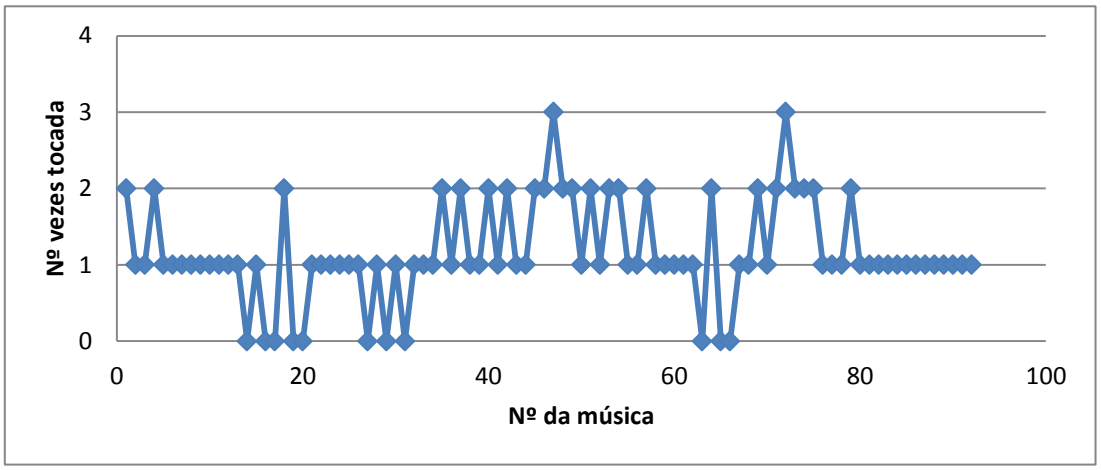



**Figura 5-4: Gráfico com a distribuição do número de vezes cada música tocou na *EcoJukeBox***

Na Figura 5-5 é apresentada uma amostra cronológica dos eventos decorridos durante os primeiros 10 minutos do primeiro dia do teste efetuado. A intensão desta cronologia de eventos é tentar perceber as decisões das músicas escolhidas para tocar com base nos eventos que foram ocorrendo durante a execução do teste.

A primeira música que tocou foi "Drifting" do artista "Andy McKee". Esta música fazia parte de um grupo de músicas adicionadas previamente à pasta partilhada da *EcoJukeBox* para que aquando do começo dos testes, fosse possível escolher uma música para começar a tocar. Como nenhuma interação tinha sido feita anteriormente, todas as músicas não tinham qualquer pontuação associada, ou seja, todas tinham uma pontuação igual. Quando existem várias músicas com melhor pontuação, escolhe-se uma música aleatoriamente desse grupo.

A segunda música tocada foi "Every Teardrop Is a Waterfall" dos "Coldplay". Esta escolha deveu-se a um voto no álbum "Mylo Xyloto", do qual a música acima faz parte. Como até ao momento da escolha da segunda música não tinham sido registados eventos com maior peso sobre a pontuação de uma música, essa foi a decisão efetuada.

A terceira música escolhida para tocar foi "Highway to Hell" dos "AC/DC". Esta música tinha sido adicionada à pasta partilhada após a escolha da música anterior para tocar. Como o critério com maior peso no perfil de grupo definido para este teste dá preferência a novas músicas adicionadas, a escolha recaiu sobre esta música.

A quarta música tocada na *EcoJukeBox* foi "Perfect Simmetry" dos "Keane". Durante o tempo em que a terceira música tocou, foram adicionadas várias novas músicas à pasta partilhada. De acordo com o critério com maior peso no perfil deste grupo, que dá preferência a músicas adicionadas recentemente, qualquer uma delas seria candidata a

tocar de seguida. O que levou à escolha desta música dos "Keane" foi uma votação no álbum "Perfect Simmetry" que fez com que a pontuação desta música fosse superior às restantes músicas adicionadas.

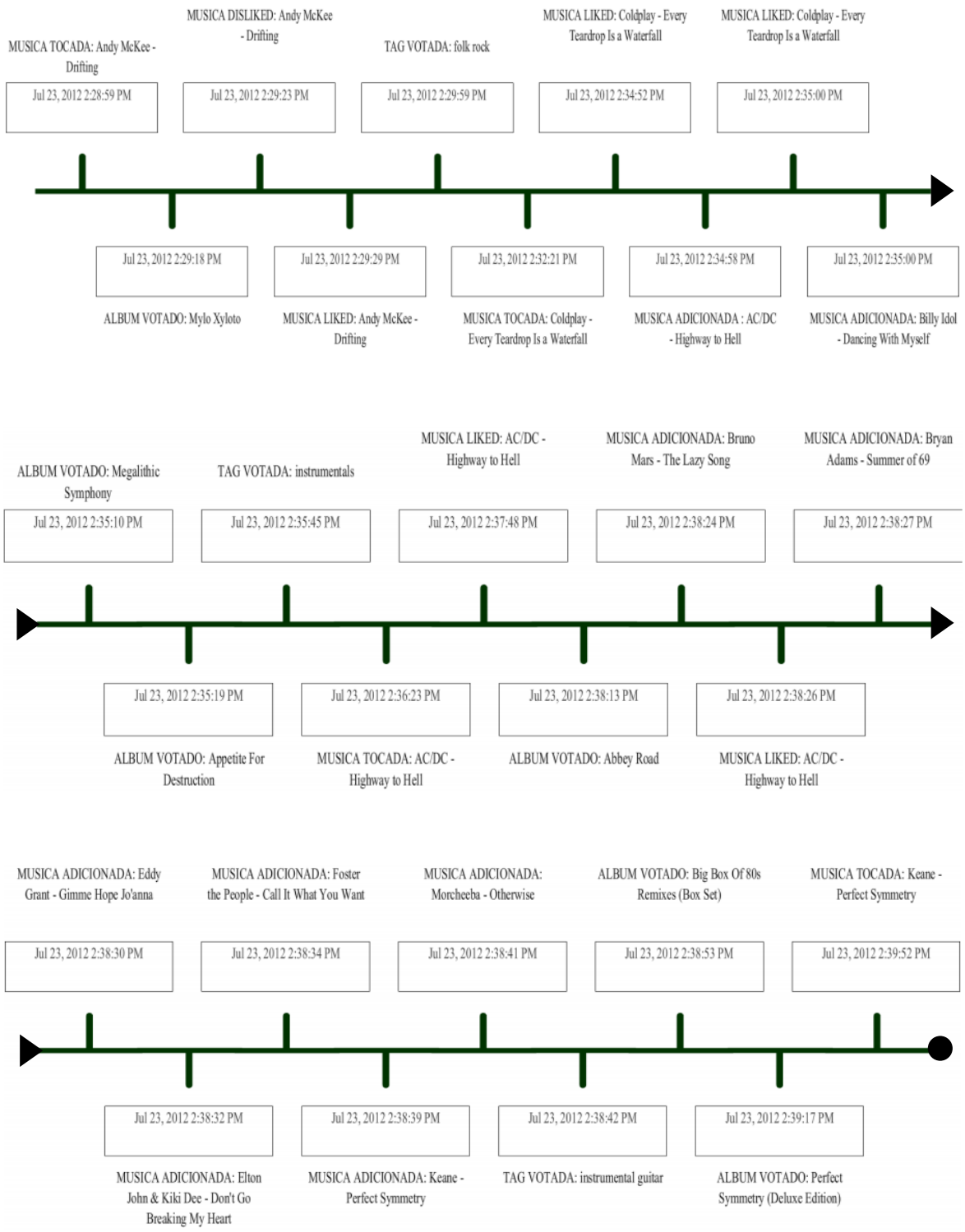



**Figura 5-5: Amostra cronológica dos eventos decorridos durante os primeiros 10 minutos na *EcoJukeBox***

## 5.2 Resultados do questionário

Todos os utilizadores participantes neste teste têm uma sólida experiência com Tecnologias de Informação e usam frequentemente outro *software* de reprodução de música (*iTunes, Winamp, Windows Media Player, Last.Fm*, etc). Numa semana normal, 4 dos 5 inquiridos utiliza este género de aplicações mais de 15 horas. O outro inquirido utiliza este tipo de aplicações entre 10 e 15 horas.

Em resposta à pergunta "Quais os fatores que são mais importantes para uma boa experiência de música em grupo" verificaram-se várias respostas: 3 utilizadores concordaram que as músicas devem ser diversificadas; 2 responderam que boas transições entre as músicas também seriam um fator importante; 2 indicaram a descoberta de novas músicas e apenas 1 indicou o fato de as músicas serem escolhidas por ele como um fator importante.

Quanto ao grau de satisfação em relação ao interface da aplicação, duas pessoas classificaram como "muito bom" e três pessoas classificaram como "bom". Relativamente ao grau de satisfação em relação ao processo de escolha de músicas, quatro pessoas classificaram como "muito bom" e apenas uma classificou como "bom".

## 5.3 Discussão em grupo

De um maneira geral, foram identificados vários aspetos positivos através da resposta dos utilizadores ao longo dos três dias de testes da *EcoJukeBox*. Os utilizadores mencionaram ter apreciado a forma como os testes foram planeados e poderem dar o seu contributo com as fases de avaliação, questionários e com a própria discussão final.

Foi referido que o prazer em utilizar este tipo de sistemas está diretamente ligado ao nível de participação das pessoas envolvidas, sendo que foi mais agradável utilizar este sistema quando eram tocadas mais músicas novas na *EcoJukeBox*. Partilhar as suas próprias músicas e depois ouvi-las em conjunto com as outras pessoas foi extremamente divertido. Os membros do grupo reconheceram que a definição do perfil de grupo é um momento fulcral para o sucesso das músicas que irão tocar no sistema.

Foram ainda dadas algumas sugestões para melhorar a experiência da *EcoJukeBox* tais como: permitir mudar de música sem que esta acabe, se houver um número considerável de pessoas que não gostem do que estão a ouvir; adicionar um novo tipo de votação

sobre os artistas das músicas; mostrar na interface *Web* estatísticas com as músicas com melhor pontuação e implementação de um sistema de *chat*.

Este teste foi importante para perceber até que ponto a *EcoJukeBox* foi de encontro das expetativas dos utilizadores ao usar um sistema deste género. Foi importante para identificar alguns aspetos que podem ser melhorados, com as sugestões dadas pelos utilizadores e que podem melhorar substancialmente a experiência de utilização deste sistema. No entanto, um dos problemas deste teste foi o facto de ter sido feito para um grupo reduzido de pessoas num espaço de tempo também reduzido. Como tal, pensamos não ser sensato tirar todas as conclusões e todas as ilações sem que outros testes possam ser efetuados para complementar estes resultados. É expectável que com um grupo de utilizadores maior e noutro tipo de ambiente de utilização os resultados possam ser manifestamente diferentes dos obtidos. O uso das anotações, que foi pouco utilizado neste teste, pode vir a ser bastante utilizado caso o repositório de músicas seja bastante maior e a pesquisa de uma determinada música possa ser bem mais complicada. As anotações são formas mais abrangentes de identificar uma música e como tal, acreditamos que possam ser o ponto-chave para uma utilização em massa.

# Capítulo 6

# Discussão e Trabalho Futuro

Neste trabalho pretendeu-se desenvolver uma solução para ambientes de partilha de música em grupo, usando algumas ideias já desenvolvidas noutros projetos e tentando também acrescentar algumas funcionalidades e melhoramentos.

Começou-se por criar o conceito de espaço comum de partilha de música, onde todos os utilizadores pudessem colocar os seus ficheiros *.mp3* preferidos. Foi construída uma interface *Web* que por um lado permite ouvir a música da *EcoJukeBox* e, por outro, permite uma forma de interação com o sistema, para influenciar a escolha das músicas e cativar as pessoas na sua utilização. O algoritmo da escolha de músicas é definido recorrendo a preferências e a exigências comuns ao grupo de utilizadores, para que o processo de escolha das músicas vá de encontro aos gostos musicais da maioria das pessoas, permitindo assim um sistema que garanta maior concordância e satisfação entre a comunidade.

A informação é persistida numa base de dados, que contém informações acerca de músicas e acerca dos acontecimentos que ocorrem da interação com os utilizadores, baseando-se nessa informação para calcular a pontuação das músicas.

O sistema invoca vários serviços externos para complementar o conhecimento musical das músicas adicionadas, recorrendo à *Last.Fm*, principalmente para obter informação relativa a anotações sobre as músicas.

Foram utilizadas várias linguagens de programação e tecnologias no desenvolvimento do projeto, tais como: *Java, C#, PHP, JavaScript, XML, JSON, Glassfish, JPA, Mysql Server, PHPmyAdmin, Servidor Apache* recorrendo sempre aos bons hábitos de programação ditados pela Engenharia de *Software* e aplicando sempre que possível os vários padrões de desenho (*Command, Singleton, Facade, Strategy, Mapper, DAO*) na sua implementação.

Os resultados dos testes apontam para um bom grau de satisfação por parte das pessoas que utilizaram a *EcoJukeBox*, que classificaram em vários aspetos o sistema com nota muito positiva. Dos resultados podemos também tirar outro tipo de conclusões, como as diferentes formas de votação existentes nesta interface, umas mais utilizadas que outras.

Quanto ao trabalho futuro, existem vários aspetos a melhorar neste projeto, com vista a torna-lo um sistema mais abrangente e completo.

Uma forma de melhorar o mesmo poderá passar pelo desenvolvimento de um sistema próprio de partilha de música, ao invés de utilizar o serviço *SugarSync*, para poder adaptar este mecanismo de partilha às necessidades da *EcoJukeBox*. Como tal, poderia ser possível associar o *upload* de músicas a cada utilizador e essa informação persistida na base de dados. Assim, cada música passaria a estar associada a um utilizador o que seria um ponto de partida para outro parâmetro de influência na escolha das músicas.

Outra sugestão de trabalho futuro poderá ser a elaboração de uma opção que permita transitar para uma nova música sem que a atual acabe de tocar, caso esta não seja do agrado da maioria dos participantes. Esta nova funcionalidade possibilitará colmatar a frustração de ouvir uma música que não é do contentamento da maior parte do público-alvo e, como tal, não faz sentido que continue a tocar. Para isso, pode ser implementada na interface *Web* uma opção em que os utilizadores possam votar para passar de música imediatamente. Para cada grupo, poderá também ser definida uma percentagem de utilizadores diferente que terão de votar na passagem imediata da música, para que tal ocorra.

Uma mais valia para este projeto seria a alteração da interface *Web*, principalmente em termos de *design*, pois apenas foi implementada uma versão protótipo que permitisse usar todas as funcionalidades da *EcoJukeBox*. Também ainda em relação à interface *Web*, podem ser adicionadas algumas estatísticas sobre as músicas (*Top* músicas mais votadas, *Top* Anotações mais votadas, *Top* Álbuns mais votados, *Top* Artistas mais votados). Foi também sugerido pelos utilizadores a implementação de um sistema de *chat* para uma maior interação entre os utilizadores e foi igualmente sugerida uma nova funcionalidade que permita efetuar votações em artistas.

A implementação de novos critérios é outro aspeto que pode ser considerado, permitindo assim construir uma elevada gama de critérios que consigam ir cada vez mais ao encontro dos gostos musicais dos utilizadores. Também poderia ser

desenvolvido um processo *online* para a definição do perfil, que pudesse evoluir ao longo do tempo mediante o desejo dos utilizadores do sistema. Nesse sistema poderiam também ser acrescentados novos critérios por parte dos utilizadores.

Outro melhoramento a desenvolver passaria por proceder a uma validação noutros cenários a fim de se perceber melhor o impacto da *EcoJukeBox*, pois infelizmente só foi possível fazer o teste num cenário muito específico, em poucos dias e para um número reduzido de participantes.

Outro dos objetivos futuros deste trabalho passa por implementar e testar a aplicação em dispositivos móveis, para que pudesse ser acedida em espaços públicos por todos os utilizadores. Para tal, uma possibilidade seria o recurso à tecnologia de *Bluetooth*.

A implementação da *EcoJukeBox* como um sistema distribuído, que permita a definição de vários grupos de utilizadores e vários espaços onde o sistema possa ser usado simultaneamente é outro caminho que seria interessante explorar no seguimento deste trabalho.

Outro melhoramento que ainda decorre da ideia da identificação de cada pessoa interveniente na *EcoJukeBox*, passaria por implementar novas formas de decisão centradas nos utilizadores, podendo-se votar também nos mesmos e influenciando assim as músicas que estão associadas a cada pessoa.

O *player* desenvolvido em *C#*, poderia ser convertido para Java para que toda a parte não aplicacional do sistema fosse desenvolvida na mesma linguagem de programação.

Outra melhoramento neste trabalho, passaria por tentar novas formas de decisão mais descentralizadas na escolha das músicas, ao contrário do trabalho que foi desenvolvido neste projeto, comparando depois os resultados obtidos em cada um dos casos.

Por último, seria importante estudar várias formas de otimizar as atualizações das pontuações das músicas baseadas nos eventos ocorridos.

# Apêndice A

# Modelo de Classes

Para organizar o sistema e garantir a sua modularidade, foi definida uma estrutura de pacotes de acordo com os vários módulos implementados:

- jukebox.config – Contém os dados de configuração da *Jukebox*. Permite definir a chave de acesso à *Last.Fm* e a localização física das músicas no servidor. Este pacote é também responsável por escrever a informação sobre os eventos que sucedem na *Jukebox* para um ficheiro de *log*.
- jukebox.domain – Contém a classe Jukebox, que permite estabelecer uma ligação à base de dados e inicializar os controladores.
- jukebox.domain.controllers – Controladores dos vários casos de uso do sistema.
- jukebox.domain.criteria – Classes que representam os vários critérios que podem ser aplicados à escolha das músicas.
- jukebox.domain.dao – Pacote responsável pelas operações efetuadas na base de dados. Possui as operações básicas, como o *add*, *remove* e *update* às tabelas.
- jukebox.domain.entity – Classes que representam as tabelas da base de dados do sistema.
- jukebox.domain.events – Representa os eventos que podem ocorrer na *Jukebox* e guarda a informação de como e quando devem ser tratados.
- jukebox.webservices – Permite exportar as funcionalidade do sistema através de serviços *Web* SOAP.
- serv.lastfm – Permite chamar serviços da Last.fm para complementar informação acerca das músicas da Jukebox.

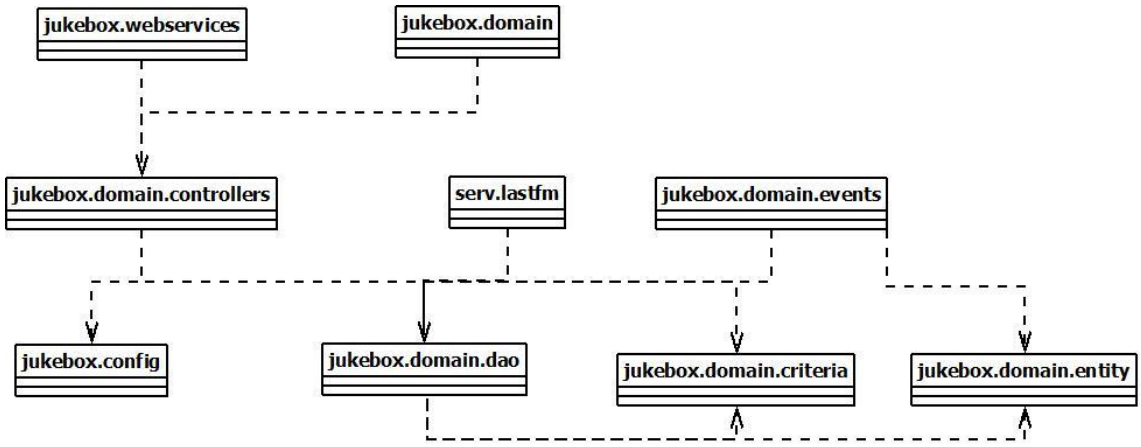



**Figura A.1: Relacionamentos dos diferentes pacotes do sistema**

| Classe | Descrição |
|---|---|
| **ConfigJukebox** | Configuração da Jukebox. |
| **Log** | Preenche um ficheiro com todas as atividades que ocorrem na Jukebox |
| **Util** | Alguns métodos auxiliares |

**Tabela A.1: Descrição das classes do pacote jukebox.config**

| Classe | Descrição |
|---|---|
| **Jukebox** | Permite estabelecer uma ligação à base de dados e inicializar todos os controladores |

**Tabela A.2: Descrição das classes do pacote jukebox.domain**

| Classe | Descrição |
|---|---|
| **ControllerAlbum** | Controlador de suporte às operações sobre os álbuns |
| **ControllerMusics** | Controlador de suporte às operações sobre as músicas |
| **ControllerPlayerIcecast** | Controlador de suporte às operações sobre o Player |
| **ControllerPoll** | Controlador de suporte às operações sobre as votações |
| **ControllerProfile** | Controlador de suporte às operações sobre os critérios |
| **ControllerTags** | Controlador de suporte às operações sobre as anotações |

**Tabela A.3: Descrição das classes do pacote jukebox.domain.controllers**

| Classe | Descrição |
|---|---|
| **Criteria** | Representa um critério sobre uma música |
| **hasArtist** | Tem um determinado artista |
| **hasTag** | Tem uma determinada anotação |
| **neverPlayed** | Nunca ter sido tocada |
| **noArtist** | Não tem um determinado artista |
| **noTag** | Não tem uma determinada anotação |
| **numDislikes** | Classificar de acordo com o número de não gostos |
| **numLikes** | Classificar de acordo com o número de gostos |
| **numLikesAlbum** | Classificar de acordo com o número de gostos nos álbuns |
| **numLikesTag** | Classificar de acordo com o número de gostos nas anotações |
| **olderAdded** | Ter sido adicionada há muito tempo |
| **olderPlayed** | Ter sido tocada há muito tempo |
| **Profile** | Representa um perfil, com uma quantidade de critérios que podem ser preferências ou exigências |
| **recentlyAdded** | Ter sido adicionada recentemente |
| **recentlyPlayed** | Ter sido tocada recentemente |

**Tabela A.4: Descrição das classes do pacote jukebox.domain.criteria**

| Classe | Descrição |
|---|---|
| **AlbumDAO** | |
| **DAO**<E> | Algumas operações simples com a base de dados |
| **DAOConnect** | Criação do Entity Manager Factory e ligação com a base de dados |
| **MusicDAO** | Algumas operações sobre a tabela music da base de dados |
| **PollDAO** | Algumas operações sobre a tabela poll da base de dados |
| **ProfileDAO** | Algumas operações sobre a tabela profile da base de dados |
| **TagDAO** | Algumas operações sobre a tabela tag da base de dados |

**Tabela A.5: Descrição das classes do pacote jukebox.domain.dao**

| Classe | Descrição |
|---|---|
| **Album** | Entidade álbum da base de dados, que representa os álbuns associados as músicas e os seus respetivos atributos |
| **Music** | Entidade music da base de dados, que representa as músicas e os seus respetivos atributos |
| **Poll** | Entidade poll da base de dados, que representa as votações das músicas e os seus respetivos atributos |
| **TagM** | Entidade tagM da base de dados, que representa as anotações associadas as músicas e os seus respetivos atributos |

**Tabela A.6: Descrição das classes do pacote jukebox.domain.entity**

| Classe | Descrição |
|---|---|
| **EventQueue** | Representa a lista de eventos em espera |
| **ReadyEvent** | Representa os eventos que estão disponíveis para serem executados assim que a ação é feita<br>Ex: Musica Tocada, Musica Votada, Álbum Votado, etc. |

**Tabela A.7: Descrição das classes do pacote jukebox.domain.events**

| Class | Description |
|---|---|
| **LastFMTags** | Classe que permite ir obter os dados da LastFm sobre as anotações das músicas |
| **LastFMTrack** | Algumas operações à API da LastFm |

**Tabela A.8: Descrição das classes do pacote serv.lastfm**

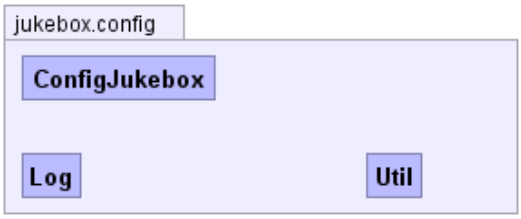



**Figura A.2: Modelo de classes do pacote jukebox.config**

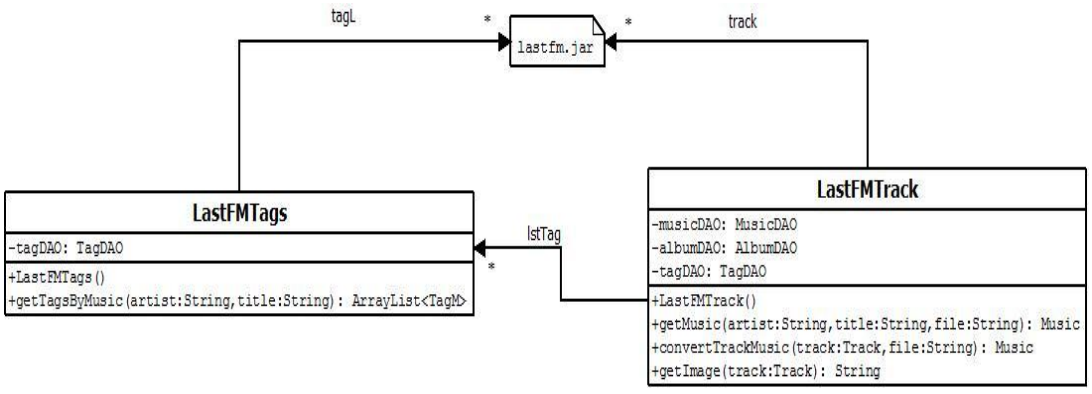



**Figura A.3: Modelo de classes do pacote serv.lastfm**

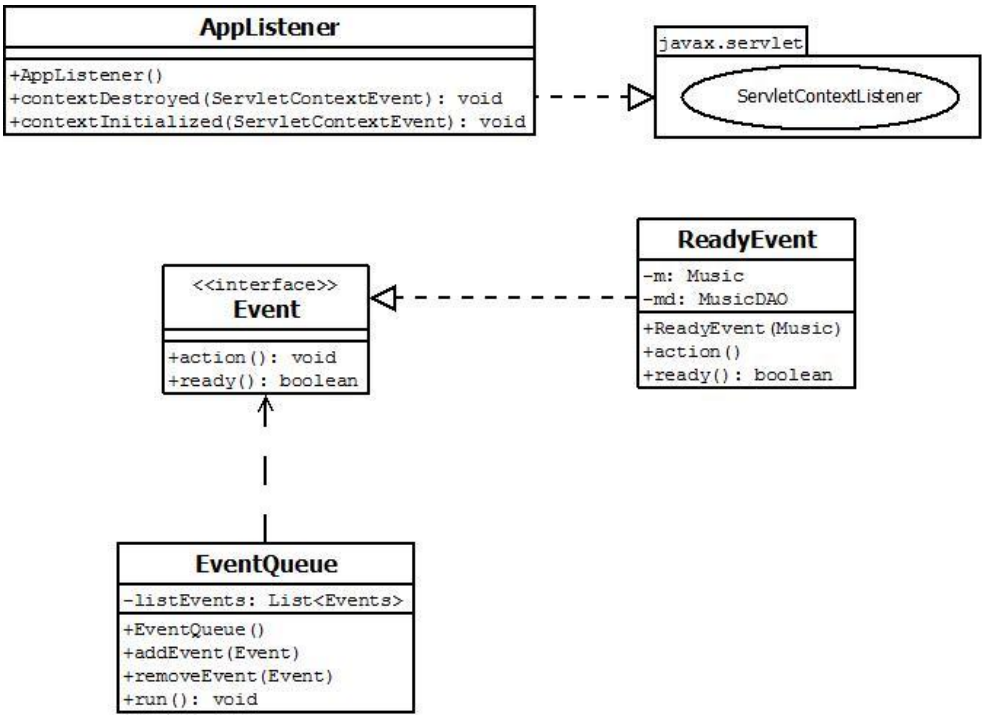



**Figura A.4: Modelo de classes do pacote jukebox.domain.events**

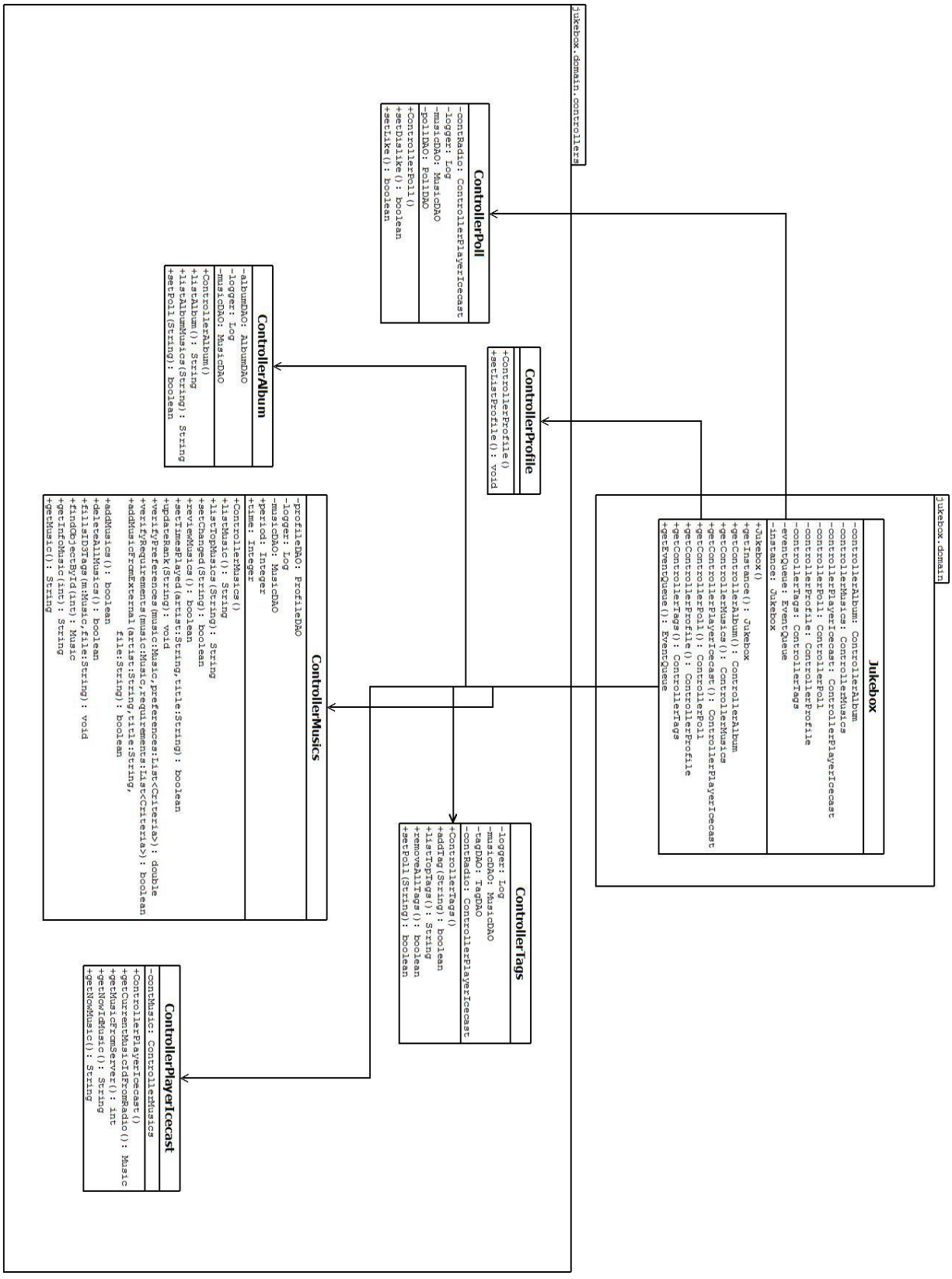



**Figura A.5: Modelo de classes do pacote jukebox.domain.controllers**

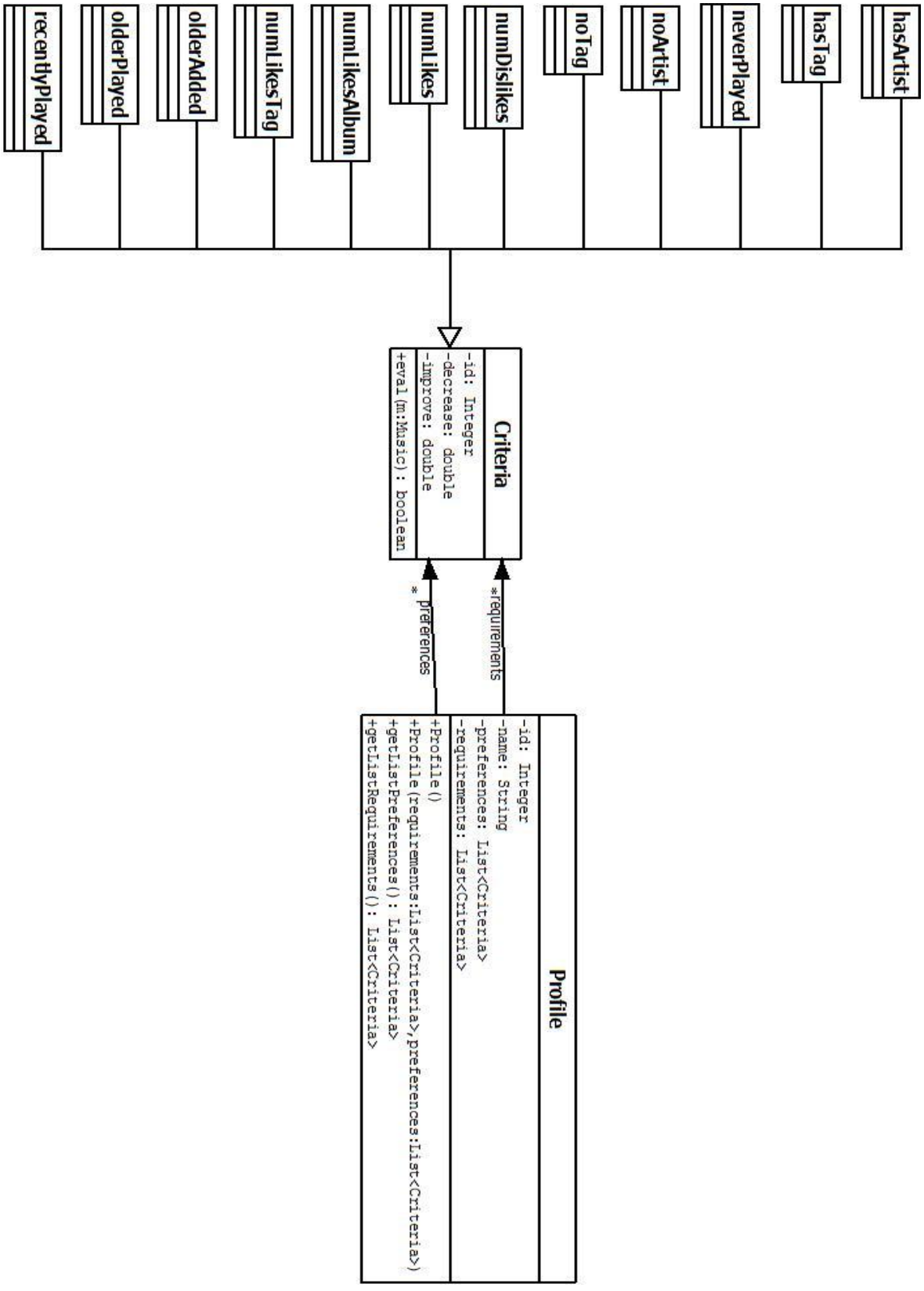



**Figura A.6: Modelo de classes do pacote jukebox.domain.criteria**

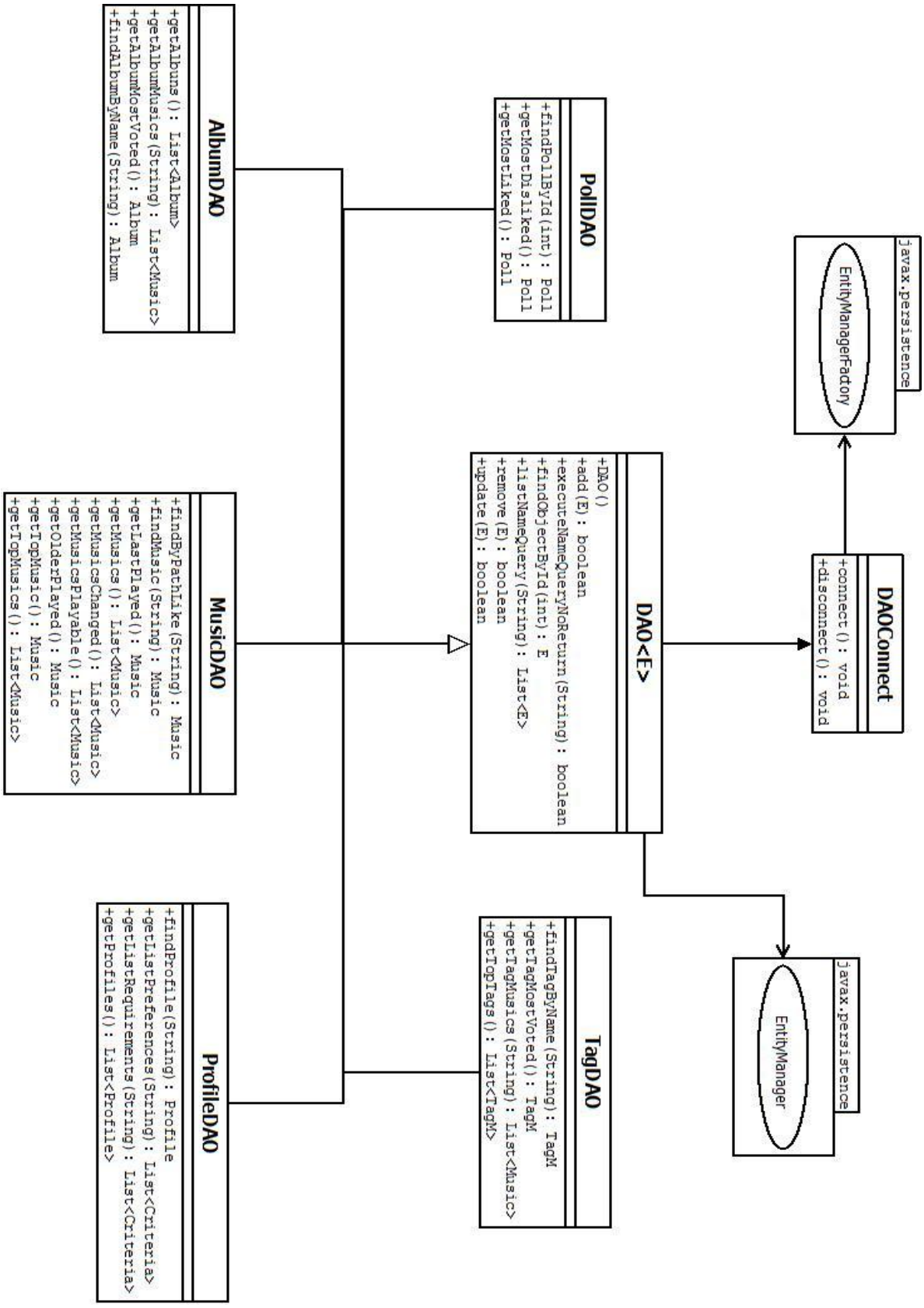



**Figura A.7: Modelo de classes do pacote jukebox.domain.dao**

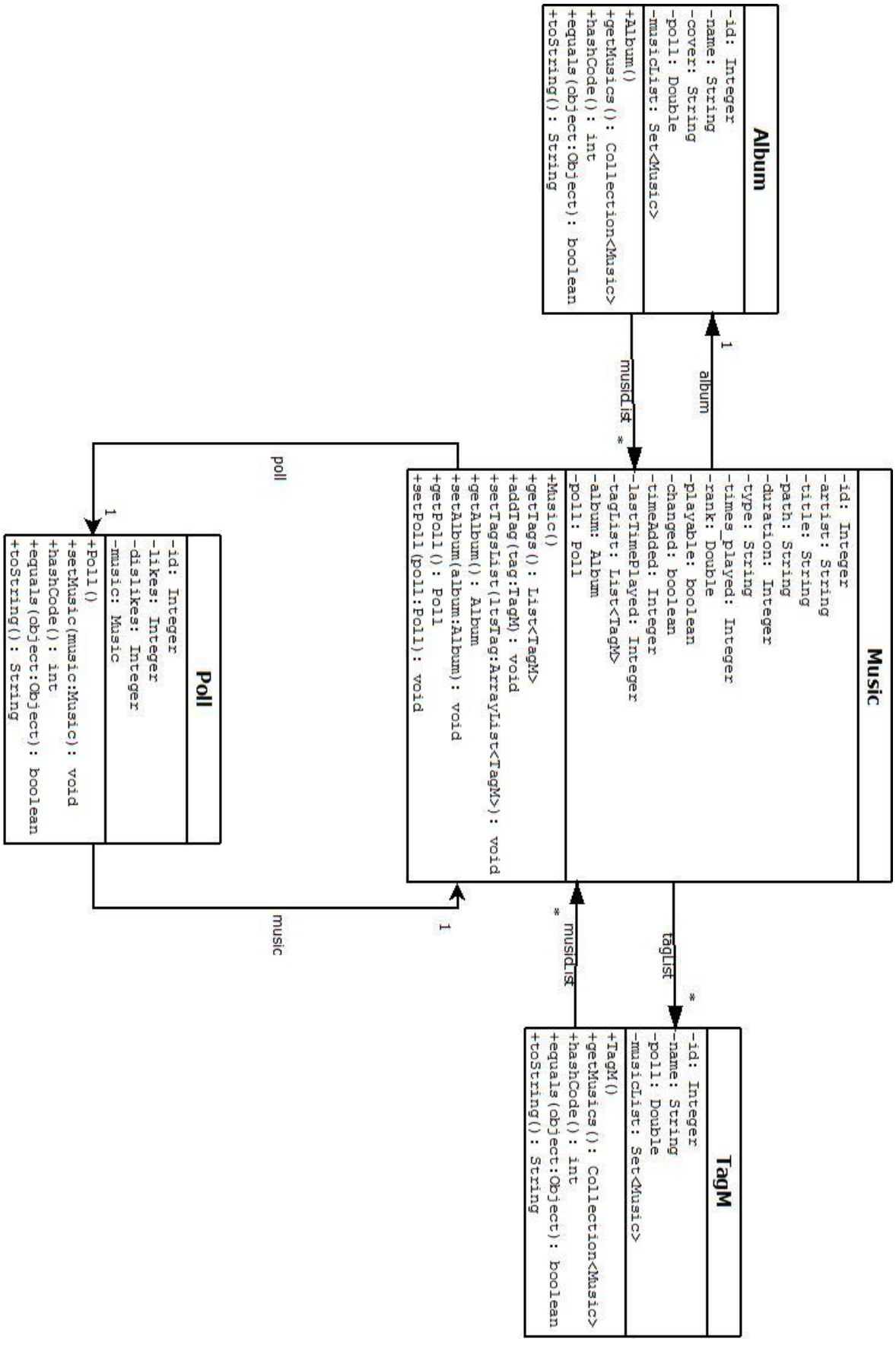



**Figura A.8: Modelo de classes do pacote jukebox.domain.entity**

# Apêndice B

# Serviços *Web* SOAP

| Nome | Descrição |
|---|---|
| listAlbumMusicJSON | Obtém a lista de músicas de um álbum serializadas |
| listAlbunsJSON | Obtém a lista de álbuns serializados |
| poll | Permite votar num álbum duma música |

**Tabela B.1: Lista de serviços na web SOAP – AlbumSOAP**

| Nome | Descrição |
|---|---|
| createListProfile | Permite construir um novo perfil, com vários critérios associados a uma lista de preferências e exigências |

**Tabela B.2: Lista de serviços na web SOAP - CriteriaSOAP**

| Nome | Descrição |
|---|---|
| addMusics | Adiciona novas músicas à EcoJukeBox |
| deleteAllMusics | Apaga todas as músicas |
| getInfoMusic | Obtém a informação da musica dado o seu *id* |
| getMusic | Obtém a música mais votada |
| listMusics | Obtém a lista de músicas |
| listTopMusics | Obtém a lista de músicas mais votadas |
| reviewMusics | Atualiza as músicas periodicamente |

**Tabela B.3: Lista de serviços na web SOAP - MusicSOAP**

| Nome | Descrição |
|---|---|
| getNowIdMusic | Obtém o *id* da música que esta a tocar |
| getNowMusic | Obtém a música que esta a tocar |

**Tabela B.4: Lista de serviços na web SOAP - PlayerSOAP**

| Nome | Descrição |
|---|---|
| dislike | Vota negativamente na música que está a tocar |
| like | Vota positivamente na música que está a tocar |

**Tabela B.5: Lista de serviços na web SOAP - PollSOAP**

| Nome | Descrição |
|---|---|
| addTag | Adiciona uma anotação à música atual |
| listTopTags | Obtém a lista de anotações mais votadas |
| poll | Vota numa anotação |
| removeAllTags | Remove todas as anotações |

**Tabela B.6: Lista de serviços na web SOAP - TagSOAP**

# Apêndice C

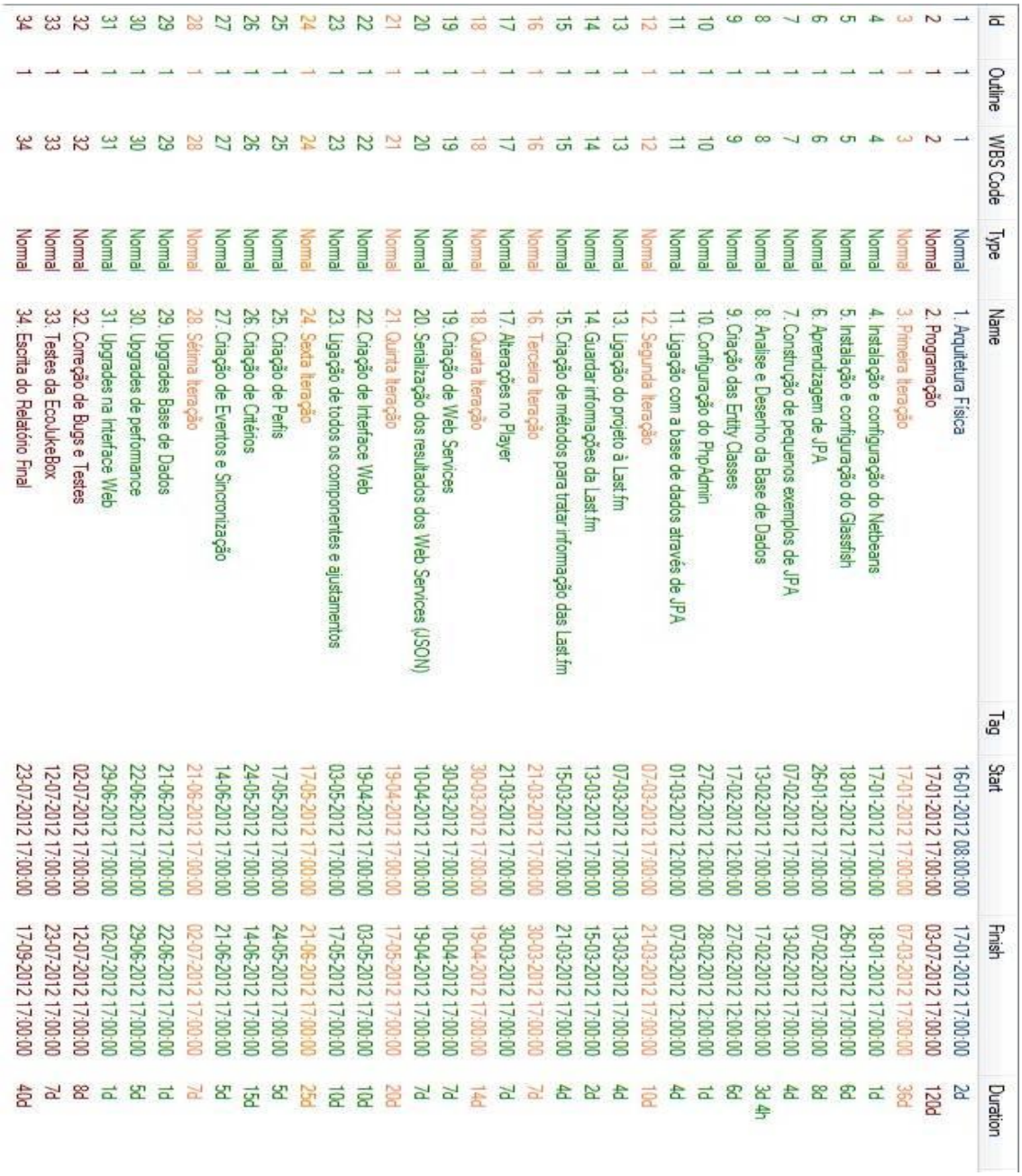
# Interface

A aplicação *Web* da *EcoJukeBox* está organizada nos seguintes ficheiros/pastas:

- jukebox.html - Página principal da *EcoJukeBox*.
- soap.php - Permite invocar os serviços SOAP.
- js/ - Pasta com os ficheiros *JavaScript*.
- images/ - Pasta com as imagens utilizadas na interface.
- css/- Pasta com os ficheiros que definem os estilos no desenho da interface.

# Apêndice D

# Mapa de Gantt



| Id | Outline | WBS Code | Type | Name | Tag | Start | Finish | Duration |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | Normal | 1. Arquitetura Física | | 16-01-2012 08:00:00 | 17-01-2012 17:00:00 | 2d |
| 2 | 1 | 2 | Normal | 2. Programação | | 17-01-2012 17:00:00 | 03-07-2012 17:00:00 | 120d |
| 3 | 1 | 3 | Normal | 3. Primeira Iteração | | 17-01-2012 17:00:00 | 07-03-2012 17:00:00 | 36d |
| 4 | 1 | 4 | Normal | 4. Instalação e configuração do Netbeans | | 17-01-2012 17:00:00 | 18-01-2012 17:00:00 | 1d |
| 5 | 1 | 5 | Normal | 5. Instalação e configuração do Glassfish | | 18-01-2012 17:00:00 | 26-01-2012 17:00:00 | 6d |
| 6 | 1 | 6 | Normal | 6. Aprendizagem de JPA | | 26-01-2012 17:00:00 | 07-02-2012 17:00:00 | 8d |
| 7 | 1 | 7 | Normal | 7. Construção de pequenos exemplos de JPA | | 07-02-2012 17:00:00 | 13-02-2012 17:00:00 | 4d |
| 8 | 1 | 8 | Normal | 8. Analise e Desenho da Base de Dados | | 13-02-2012 17:00:00 | 17-02-2012 12:00:00 | 3d 4h |
| 9 | 1 | 9 | Normal | 9. Criação das Entity Classes | | 17-02-2012 12:00:00 | 27-02-2012 12:00:00 | 6d |
| 10 | 1 | 10 | Normal | 10. Configuração do PhpAdmin | | 27-02-2012 12:00:00 | 28-02-2012 12:00:00 | 1d |
| 11 | 1 | 11 | Normal | 11. Ligação com a base de dados através de JPA | | 01-03-2012 12:00:00 | 07-03-2012 12:00:00 | 4d |
| 12 | 1 | 12 | Normal | 12. Segunda Iteração | | 07-03-2012 17:00:00 | 21-03-2012 17:00:00 | 10d |
| 13 | 1 | 13 | Normal | 13. Ligação do projeto à Last.fm | | 07-03-2012 17:00:00 | 13-03-2012 17:00:00 | 4d |
| 14 | 1 | 14 | Normal | 14. Guardar informações da Last.fm | | 13-03-2012 17:00:00 | 15-03-2012 17:00:00 | 2d |
| 15 | 1 | 15 | Normal | 15. Criação de métodos para tratar informação das Last.fm | | 15-03-2012 17:00:00 | 21-03-2012 17:00:00 | 4d |
| 16 | 1 | 16 | Normal | 16. Terceira Iteração | | 21-03-2012 17:00:00 | 30-03-2012 17:00:00 | 7d |
| 17 | 1 | 17 | Normal | 17. Alterações no Player | | 21-03-2012 17:00:00 | 30-03-2012 17:00:00 | 7d |
| 18 | 1 | 18 | Normal | 18. Quarta Iteração | | 30-03-2012 17:00:00 | 19-04-2012 17:00:00 | 14d |
| 19 | 1 | 19 | Normal | 19. Criação de Web Services | | 30-03-2012 17:00:00 | 10-04-2012 17:00:00 | 7d |
| 20 | 1 | 20 | Normal | 20. Serialização dos resultados dos Web Services (JSON) | | 10-04-2012 17:00:00 | 19-04-2012 17:00:00 | 7d |
| 21 | 1 | 21 | Normal | 21. Quinta Iteração | | 19-04-2012 17:00:00 | 17-05-2012 17:00:00 | 20d |
| 22 | 1 | 22 | Normal | 22. Criação de Interface Web | | 19-04-2012 17:00:00 | 03-05-2012 17:00:00 | 10d |
| 23 | 1 | 23 | Normal | 23. Ligação de todos os componentes e ajustamentos | | 03-05-2012 17:00:00 | 17-05-2012 17:00:00 | 10d |
| 24 | 1 | 24 | Normal | 24. Sexta Iteração | | 17-05-2012 17:00:00 | 21-06-2012 17:00:00 | 25d |
| 25 | 1 | 25 | Normal | 25. Criação de Perfis | | 17-05-2012 17:00:00 | 24-05-2012 17:00:00 | 5d |
| 26 | 1 | 26 | Normal | 26. Criação de Critérios | | 24-05-2012 17:00:00 | 14-06-2012 17:00:00 | 15d |
| 27 | 1 | 27 | Normal | 27. Criação de Eventos e Sincronização | | 14-06-2012 17:00:00 | 21-06-2012 17:00:00 | 5d |
| 28 | 1 | 28 | Normal | 28. Sétima Iteração | | 21-06-2012 17:00:00 | 02-07-2012 17:00:00 | 7d |
| 29 | 1 | 29 | Normal | 29. Upgrades Base de Dados | | 21-06-2012 17:00:00 | 22-06-2012 17:00:00 | 1d |
| 30 | 1 | 30 | Normal | 30. Upgrades de performance | | 22-06-2012 17:00:00 | 29-06-2012 17:00:00 | 5d |
| 31 | 1 | 31 | Normal | 31. Upgrades na Interface Web | | 29-06-2012 17:00:00 | 02-07-2012 17:00:00 | 1d |
| 32 | 1 | 32 | Normal | 32. Correção de Bugs e Testes | | 02-07-2012 17:00:00 | 12-07-2012 17:00:00 | 8d |
| 33 | 1 | 33 | Normal | 33. Testes da EcoJukeBox | | 12-07-2012 17:00:00 | 23-07-2012 17:00:00 | 7d |
| 34 | 1 | 34 | Normal | 34. Escrita do Relatório Final | | 23-07-2012 17:00:00 | 17-09-2012 17:00:00 | 40d |

**Figura D.1: Tarefas do Projeto**

**Figura D.2: Mapa de Gantt**

# Apêndice E

# Resultados dos Inquéritos

Neste apêndice apresentam-se as várias questões colocadas aos utilizadores que participaram no teste do cenário simulado de utilização da *EcoJukeBox* e uma tabela com as respostas dos vários utilizadores a cada pergunta.

**Q1.** Experiência com Tecnologias de Informação
(Escala de 1 a 5)

**Q2.** Uso de outro *software* de reprodução de música (*iTunes,Winamp,Windows Media Player,Last.Fm,*etc)
(Escala de 1 a 5)

**Q3.** Número de horas que utilizam este género de aplicações por semana:
- Nunca
- Menos de 5h
- Entre 5 e 10h
- Entre 10 e 15h
- Mais de 15h

**Q4.** Quais os fatores que são mais importantes para uma boa experiência de música em grupo:
- Muitas das músicas serem sugeridas por mim
- As músicas serem diversificadas
- Descobrir novas músicas
- Boas transições entre as músicas

**Q5.** Qual o grau de satisfação em relação ao seu interface da aplicação?
(Escala de 1 a 5)

**Q6.** Qual o grau de satisfação em relação ao processo de escolha de músicas? (Escala de 1 a 5)

| | User1 | User2 | User3 | User4 | User5 |
|---|---|---|---|---|---|
| **Q1** | 5 | 5 | 5 | 5 | 5 |
| **Q2** | 5 | 5 | 5 | 5 | 5 |
| **Q3** | +15h | +15h | 10h - 15h | +15h | +15h |
| **Q4** | 2, 3 | 2, 4 | 2 | 1, 4 | 3 |
| **Q5** | 4 | 5 | 4 | 5 | 4 |
| **Q6** | 5 | 5 | 5 | 5 | 4 |

**Tabela E.1: Resultados do questionário de utilização da *EcoJukeBox***

# Apêndice F

# Modelo Conceptual

Neste apêndice apresenta-se o modelo conceptual da base de dados, representando as várias entidades que irão foram persistidas. Podemos dividir a base de dados em dois grupos distintos: música e perfis de grupo.

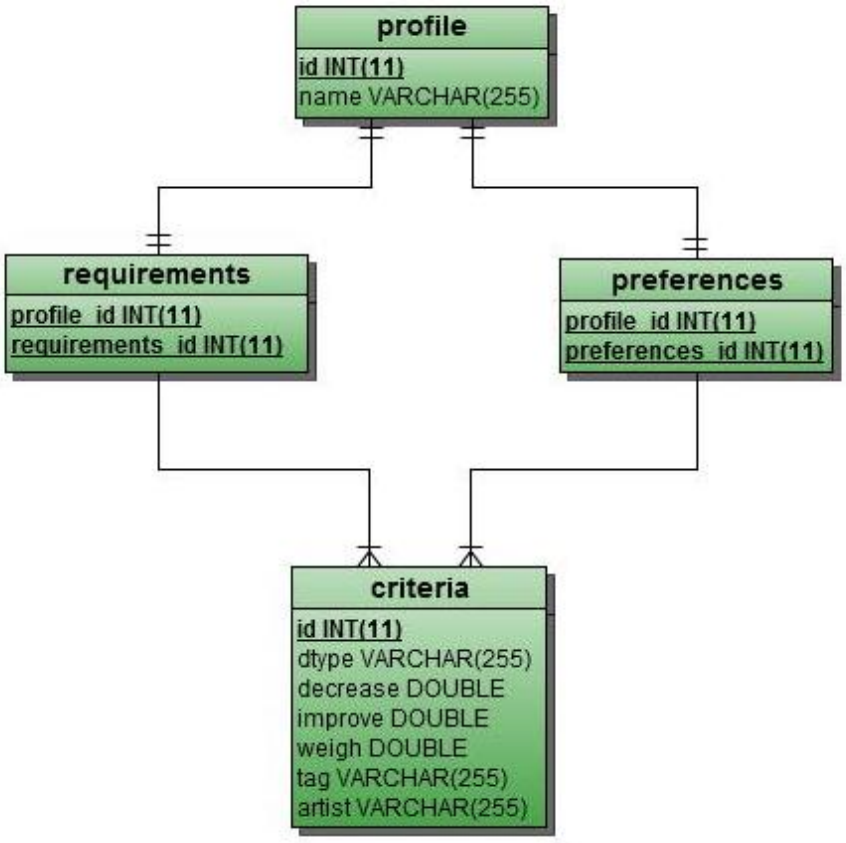



**Figura F.1: Modelo conceptual da base de dados (tabelas relativas aos perfis de grupo)**

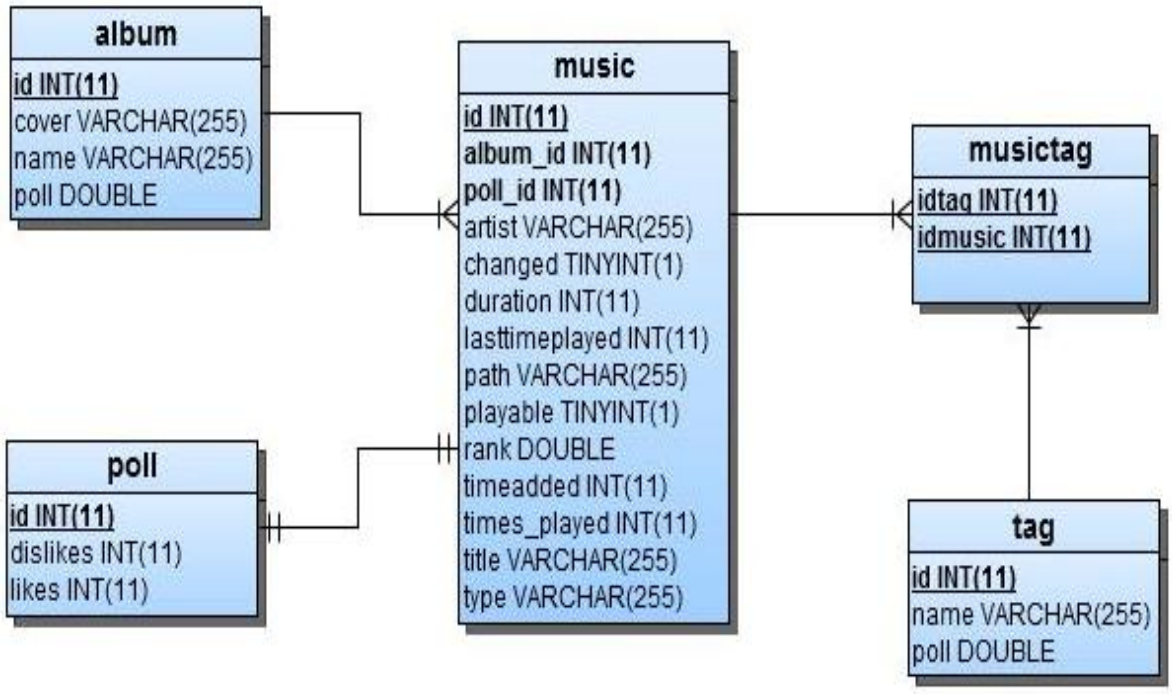



**Figura F.2: Modelo conceptual da base de dados (tabelas relativas às músicas)**

# Apêndice G

# Glossário

***MP3* –** Criado na Alemanha, o *MPEG-1 Layer 3* é o formato de música digital mais conhecido e utilizado em todo o mundo. O esquema de compressão equivale a três camadas distintas de som, cada uma com uma finalidade diferente. Com esta compressão, há grande perda na qualidade do áudio, já que todas as faixas que teoricamente não são captadas pelo ouvido humano, são eliminadas. A taxa de compressão equivale a 10:1, sendo possível gravar um *CD* com mais de 12 horas de *MP3* a uma taxa de 128 *KBPS*.

***Java Enterprise Edition*** – É uma plataforma de programação na linguagem de programação *Java*. Difere da versão *standard* pela adição de bibliotecas que fornecem novas funcionalidades: desenvolver *software* distribuído, tolerância a falhas, baseada em componentes modulares e que contém bibliotecas para acesso a bases de dados. A plataforma *JEE* contém uma série de *containers* ou *API*s:

- *JDBC (Java Database Connectivity)*, utilizado no acesso a bases de dados;
- *Servlets*, são utilizados para o desenvolvimento de aplicações Web com conteúdo dinâmico.
- *JSP (Java Server Pages)*, uma especialização do *servlet* que permite que conteúdo dinâmico seja facilmente desenvolvido.
- *JTA (Java Transaction API)*, é uma *API* que padroniza o tratamento de transações dentro de uma aplicação *Java*.
- *EJBs (Enterprise Java Beans)*, utilizados no desenvolvimento de componentes de *software*.
- *JCA (Java Connector Architecture)*, é uma *API* que padroniza a ligação a aplicações legadas.

- *JPA (Java Persistence API)*, é uma *API* que padroniza o acesso a bases de dados através de mapeamento Objeto/Relacional dos *Enterprise Java Beans*.

***SugarSync* –** Serviço que sincroniza arquivos entre computadores e outros dispositivos para acesso a arquivos, *backup*, sincronização e partilha de uma variedade de sistemas operacionais, como *Windows*, *Mac OS X*, *iOS* (*iPhone, iPad, iPod Touch*), *Android*, *BlackBerry*, *Windows Mobile* e *Symbian*.

**C# -** Também conhecido como *C Sharp*, é uma linguagem de programação orientada a objetos, desenvolvida pela *Microsoft* como parte da plataforma *.NET*. A sua sintaxe orientada a objetos foi baseada no *C++* mas inclui muitas influências de outras linguagens de programação, como *Pascal* e *Java*.

***GlassFish* –** Servidor de aplicação desenvolvido pela *Sun Microsystems* para a plataforma *Java 2 Enterprise Edition* (*Java2EE*).

***Icecast2* –** Servidor de transmissão de áudio de código-livre que qualquer um pode modificar, usar e mexer.

***JSON* –** Permite serializar objetos complexos entre diferentes linguagens de programação. O seu tratamento tem melhor desempenho do que o *XML* devido à sua estrutura ilegível para o olho humano.

***JDBC* –** *Java Database Connectivity* ou *JDBC* é um conjunto de classes e interfaces (*API*) escritas em *Java* que fazem o envio de instruções *SQL* para qualquer base de dados relacional.

***POJO* –** são objetos *Java* que seguem um desenho simplificado.

# Apêndice H

# Fórmulas dos critérios de seleção de músicas

### *1. Tocada recentemente*

$$V = 1 - \frac{Nemissao - Vm}{Nemissao - MenorValor}$$

Legenda:

*Nemissao* - tempo atual da emissão

*Vm* - tempo em que a música tocou pela última vez

*MenorValor* - tempo da música que tocou há mais tempo

### *2. Adicionada recentemente*

$$V = 1 - \frac{Nemissao - Vm}{Nemissao - MenorValor}$$

Legenda:

*Nemissao* - tempo atual da emissão

*Vm* - tempo em que a música tocou pela última vez

*MenorValor* - tempo da música que tocou há mais tempo

### *3. Número de "gostos"*

$$V = 1 - \frac{NMgostos - Vm}{NMgostos - Nmgostos}$$

Legenda:

*NMgostos* – número maior de gostos numa música

*Vm* – número de gostos da música

*Nmgostos* – número menor de gostos numa música

### *4. Número de votos num álbum*

$$V = 1 - \frac{NMalbum - Vm}{NMalbum - Nmalbum}$$

Legenda:

*NMalbum* - número maior de gostos num álbum

*Vm* – número de gostos no álbum da música

*Nmalbum* - número menor de gostos num álbum

### *5. Número de votos numa anotação*

$$V = 1 - \frac{NManotacao - Vm}{NManotacao - Nmanotacao}$$

Legenda:

*NManotacao* - número maior de gostos numa anotação

*Vm* - número de gostos na anotação da música

*Nmanotacao* - número menor de gostos numa anotação

### *6. Número de "não gostos"*

$$V = \frac{NMngostos - Vm}{NMngostos - Nmngostos}$$

Legenda:

*NMngostos* – número maior de não gostos numa música

*Vm* – número de não gostos da música

*Nmngostos* – número menor de não gostos numa música

### *7. Não ter sido tocada recentemente*

$$V = \frac{Nemissao - Vm}{Nemissao - MenorValor}$$

Legenda:

*Nemissao* - tempo atual da emissão

*Vm* - tempo em que a música tocou pela última vez

*MenorValor* - tempo da música que tocou há mais tempo

### *8. Não ter sido adicionada recentemente*

$$V = \frac{Nemissao - Vm}{Nemissao - MenorValor}$$

Legenda:

*Nemissao* - tempo atual da emissão

*Vm* - tempo em que a música tocou pela última vez

*MenorValor* - tempo da música que tocou há mais tempo

# Bibliografia


[1] História da *Jukebox* 1888-1913

http://juke-box.dk/gert-history88-13.htm (18-08-2012)

[2] K. O'Hara, M. Lipson, M. Jansen, A. Unger, H. Jeries, and P. Macer, Jukola: democratic music choice in a public space, Proc. of DIS ‘04, pages 145-154, 2004.

[3] D. Sprague, F. Wu, M. Tory, Music selection using the PartyVote democratic jukebox, Proc. of AVI '08, pages 433-436, 2008.

[4] S. Cunningham, D. Nichols, Exploring Social Music Behavior: An Investigation of Music Selection at Parties, Proc. of ISMIR ‘09, pages 747-752, 2009.

[5] M. Jacobsson, M. Rost, M. Håkansson, L. Holmquist, Push!Music: Intelligent Music Sharing on Mobile Devices, Proc. of UbiComp ‘05, 2005.

[6] E. Martínez, Ò. Celma, M. Sordo, B. Jong, X. Serra, Extending the folksonomies of freesound.org using content-based audio analysis, Proc. of SMC ‘09, pages 65-70, 2009.

[7] M. Crampes, J. Villerd, A. Emery, S. Ranwez, Automatic Playlist Composition in a Dynamic Music Landscape, Proc. of SADPI ’07, pages 15-20, 2007.

[8] D. Baur, A. Butz, Pulling Strings from a Tangle: Visualizing a Personal Music Listening History, Proc. of IUI ‘09, pages 439-444, 2009.

[9] G. Popescu, P. Pu, What’s the Best Music You Have? Designing Music Recommendation for Group Enjoyment in GroupFun, Proc. of CHI ‘12, 2012.

[10] H. Sørensen, J. Kjeldskov, Distributed Interaction: A Multi-Device, Multi-User Music Experience, Proc. of AVI´12, pages 336-339, 2012.

[11] API da Last.fm http://www.lastfm.pt/api (18-08-2012)

[12] Comparação entre software de Sincronização de Ficheiros

http://en.wikipedia.org/wiki/Comparison_of_file_synchronization_software (18-08-2012)

[13] Chris Adamson, QuickTime for Java: A Developer's Notebook, O'Reilly Media, 2005.

[14] Mike Keith, Merrick Schincariol, PRO JPA2 Mastering the Java Persistence API, Apress, 2009.

[15] Dustin Diaz, and Ross Harmes, Pro JavaScript Design Patterns, Apress, 2007.

[16] Christopher G. Lasater, Design Patterns, Jones & Bartlett Publishers, 2006.

[17] William Sanders, Chandima Cumaranatunge, ActionScript 3.0 Design Patterns: Object Oriented Programming Techniques, Adobe Developer Library, 2007.

[18] William Crawford, Jonathan Kaplan, J2EE Design Patterns, O'Reilly Media, 2003.

[19] Eric Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI, Addison-Wesley Professional, 2002.

[20] SugarSync https://www.sugarsync.com/developer (18-08-2012)

[21] Diogo Serrano, This is a RadioBot, 2011.

[22] Martin Fowler, Patterns of Enterprise Application Architecture, 1 edition, Addison-Wesley Professional, 2002.

[23] Icecast2 http://www.icecast.org/docs/icecast-2.3.3/ (18-08-2012)

[24] Glassfish http://glassfish.java.net/docs/project.html (18-08-2012)

[25] Java Enterprise Edition http://docs.oracle.com/javaee/ (18-08-2012)

[26] C# http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx (18-08-2012)

[27] JSON http://blog.technologyofcontent.com/2010/01/json-vs-xml/ (18-08-2012)

[28] Dropbox https://www.dropbox.com/developers/reference/api (18-08-2012)

[29] Sky Drive http://msdn.microsoft.com/en-us/library/live/hh826545.aspx

(18-08-2012)

[30] SpiderOak https://spideroak.com/diy/api_docs.html (18-08-2012)

[31] Wuala http://www.wuala.com/Wuala%20API/Documentation?lang=en

(18-08-2012)

[32] iTunes http://www.apple.com/itunes/affiliates/resources/documentation.html

(18-08-2012)

[33] Windows Media Player

http://msdn.microsoft.com/en-us/library/windows/desktop/dd564679(v=vs.85).aspx (18-08-2012)

[34] Winamp http://www.winamp.com/help/Support (18-08-2012)

[35] MusicMatch http://www.iomega.com/support/manuals/hipzip/musicmatch.html (18-08-2012)

[36] Tags ID3 v2 http://www.id3.org/d3v2.3.0 (18-08-2012)